

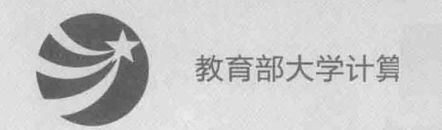
教育部大学计算机课程改革项目规划教材

Python语言程序设计基础

（第2版）

嵩天 礼欣 黄天羽 著

高等教育出版社



教育部大学计算　　　　　材

Python语言程序设计基础

（第2版）

嵩天　礼欣　黄天羽　著

高等教育出版社·北京

内容提要

本书提出了以理解和运用计算生态为目标的 Python 语言教学思想，在系统讲解 Python 语言语法的同时介绍了从数据理解到图像处理的 14 个 Python 函数库，向初学 Python 语言的读者展示了全新的编程语言学习路径。全书一共设计了 25 个非常具有现代感的实例，从绘制蟒蛇、理解天天向上的力量到机器学习、网络爬虫，从文本进度条、统计名著人物重要性到图像手绘效果、雷达图绘制，绝大多数实例为作者原创，将随着内容深入不断激发读者学习 Python 语言的热情，因为“编程是件很有趣的事儿”。

本书内容丰富、叙述清晰、循序渐进，采用新形态构建形式，提供大量扩展阅读资料、学习资料和学习视频。本书作者（中国大学 MOOC 平台“Python 语言程序设计”课程的主讲教师）建议广大读者借助在线开放课程，深入学习本书内容。本书适合初学 Python 语言的读者使用，也适合作为各类大专院校的教材，同时，也可作为对 Python 感兴趣读者的自学参考书。

图书在版编目（CIP）数据

Python 语言程序设计基础/嵩天，礼欣，黄天羽著
. --2 版. --北京：高等教育出版社，2017.2
ISBN 978-7-04-047170-0

Ⅰ. ①P… Ⅱ. ①嵩… ②礼… ③黄… Ⅲ. ①软件工具-程序设计-教材 Ⅳ. ①TP311.561

中国版本图书馆 CIP 数据核字（2016）第 324656 号

Python Yuyan Chengxu Sheji Jichu

策划编辑 刘 娟　　责任编辑 刘 娟　　封面设计 李卫青　　版式设计 马 云
插图绘制 杜晓丹　　责任校对 高 歌　　责任印制 尤 静

出版发行	高等教育出版社	网 址	http://www.hep.edu.cn
社 址	北京市西城区德外大街 4 号		http://www.hep.com.cn
邮政编码	100120	网上订购	http://www.hepmall.com.cn
印 刷	北京鑫丰华彩印有限公司		http://www.hepmall.com
开 本	850mm × 1168mm 1/16		http://www.hepmall.cn
印 张	20.75	版 次	2014 年 7 月第 1 版
字 数	450 千字		2017 年 2 月第 2 版
购书热线	010-58581118	印 次	2017 年 2 月第 2 次印刷
咨询电话	400-810-0598	定 价	39.00 元

物 料 号 47170-00

数字课程资源使用说明

与本书配套的数字课程资源发布在高等教育出版社易课程网站，请登录网站后开始课程学习。

一、注册/登录

访问 http://abook.hep.com.cn/1865445，点击“注册”，在注册页面输入用户名、密码及常用的邮箱进行注册。已注册的用户直接输入用户名和密码登录即可进入“我的课程”页面。

二、课程绑定

点击“我的课程”页面右上方“绑定课程”，正确输入教材封底防伪标签上的20位密码，点击“确定”完成课程绑定。

三、访问课程

在“正在学习”列表中选择已绑定的课程，点击“进入课程”即可浏览或下载与本书配套的课程资源。刚绑定的课程请在“申请学习”列表中选择相应课程并点击“进入课程”。

四、与本书配套的易课程数字课程资源包括案例素材，以便读者学习使用。

账号自登录之日起一年内有效，过期作废。
如有账号问题，请发邮件至：abook@hep.com.cn。



五、资源使用

与本书配套的易课程数字课程资源按照章、节知识树的形式构成，包括电子教案、MOOC课程、源代码、阶段测试、程序设计、程序练习、图片资料、彩图素材、程序素材等内容的资源，以便读者学习使用。

1. 电子教案：教师上课使用的与课程和教材紧密配套的教学PPT，可通过二维码扫描观看，以便学生课前预习或课后复习使用，也可供教师下载使用。

2. MOOC课程：提供本书编者开设在中国大学MOOC上的“Python语言程序设计”课程的外链，学生通过扫描二维码即可观看。

3. 源代码、阶段测试、程序设计、程序练习：本书所配套的与代码相关的练习，均可通过扫描书中边栏的二维码进入编者团队设计开发的Python123平台查看和运行。也可通过扫描前勒口处的二维码进入平台。

4. 图片资料：针对函数库的相关属性和知识点，本书提供快速参考索引，学生可通过扫描书中边栏的二维码获取，以便帮助学生增进理解，加深印象。

5. 彩图素材、程序素材：提供与知识点相关的彩图和程序素材。

第 1 版前言

本书是在国内外广泛关注且推进“计算思维”教学理念的大背景下编写的，在该书成稿之时，如何将“计算思维”理念转化为大学计算机基础课程的教学内容仍处于探讨中。无论“计算思维”的内涵和外延如何，具有“计算思维”的学习者应该能够深刻理解问题的计算特性并善于利用计算机解决问题。本书以此为出发点，期望实现两个目标：使读者掌握一门终身受用的程序设计语言（Python 语言）；使读者体验利用程序设计语言解决实际问题的过程和思路。

选择 Python 语言作为“终身受用的程序设计语言”来教学并非因为作者在 10 年前就接触并使用它，而是因为 Python 语言是一种简洁且强大的语言。相比其他高级语言，它的语法简洁质朴，可以用优美来形容。最关键的，它是一种开源的脚本语言，这个特点促使世界上出现了最大的围绕 Python 程序设计的开放社区。至今，该社区已经提供了超过 3 万个不同功能的开源函数库，为基于 Python 语言的快速开发提供了强大支持。

超凡脱俗、简洁优美、功能强大、跨各种平台、经济实惠等词语都可以用来形容 Python 语言，但这些还不够，Python 语言是仅次于 C 语言的第二语言。而更通常的情况是，如果程序不是以执行性能为首要设计目标，Python 语言是首选。

Python 语言是一门非常简单易学的语言。作者曾经设计过“1 小时学 Python”的教学实验，实践证明，大多数没有任何程序设计基础的大一学生都可以在 1 小时内理解 Python 设计方法并具备十几行代码的编写能力。这曾让作者既喜且忧，喜在学生们似乎找到了一种简单易学、编写快速并能解决问题的合适语言，忧在至今 Python 语言还没有进入大学计算机基础课程的教学计划。本书是一个尝试，希望更多同行关注 Python 语言，在大学计算机基础课程教学中讲授 Python 语言，让学生们终身受益。

本书具有较强的现代气息，书中所涉及的问题不仅包括房屋贷款计算，还包括 PM2.5 雾霾预警、圆周率计算、GPS 定位和科赫曲线（分形几何）等，读者在解决一个个问题的同时一定不会觉得乏味，因为这些问题就在身边。为了配合读者学习或高校教师开展 Python 语言教学，本书通过配套网站提供电子资源，网址为 http://www.python123.org，内容包括教学用 PPT、更多习题和答案、更多程序设计问题和实例、在线程序测试平台、读者在线问题解答等。

北京理工大学计算机学院李凤霞教授是本书的主审，她的直觉、睿智和丰富经验启发了

作者对教学 Python 程序设计语言的思路。还要感谢在本书撰写和出版过程中给予帮助的人，包括研究生史湘君、骆世瑛、徐金楠、万云凯、李玮、陈潇、张运大、刘翼和易琳等，以及北京理工大学教师孙新，他们对本书部分段落和例子都有实际贡献。本书得到了北京市教育委员会“北京高等学校青年英才计划项目”的资助，在此一并感谢。限于水平，书中不足之处在所难免，敬请读者和同行批评指正。作者的电子邮件地址是 songtian@bit.edu.cn。

编　者

2013年12月

第 2 版前言

——Python 语言是什么？它只是其他编程语言的替代品吗？

——不，Python 是一种生态语言。

地球几十亿年的生命孕育出丰富多彩的自然生态，物竞天择，依存繁衍。计算机 70 余年的发展开创了一个释放全球智力、激发创新热情的开源共享的信息时代。随着专业分工和智慧角逐的深入，各信息技术分支逐渐形成了一批以开源共享为形态的开放资源，包括开源操作系统、数据库、软件工具甚至开源硬件，构成了“计算生态”。与自然生态类似，计算生态并没有顶层设计，而是获益于草根工程师或一线专家无私贡献的专业智慧。计算生态中的各元素在竞争中发展、依存、终结、再生，成为信息技术快速发展最重要的创新动力。

Python 语言在计算生态的大背景下诞生、发展、再生，历时近 30 年，其简洁和面向生态的设计理念得到了广泛认同，形成了全球范围最大的单一语言编程社区。超过 9 万个第三方编程库覆盖从数据到智能、二维到三维、文本处理到虚拟现实、控制逻辑到系统结构等几乎所有的计算领域。最为可贵的是，Python 语言能够将其他编程语言的优秀成果封装起来，降低使用复杂度。因此，我们称 Python 语言为“生态语言”。

本书在国内高校广泛接触并关注 Python 语言教学的大背景下编写，试图从计算技术发展角度阐释 Python 语言作为“生态语言”的价值，展示一条与其他编程语言不同的学习路径。具体来说，本书设计了超过 20 个利用第三方库的编程实例，伴随 Python 语言语法讲解了 10 余个标准库或第三方库的使用，在讲解程序设计基础概念、Python 语言语法的同时，帮助读者理解围绕计算生态开展编程并解决问题的基本理念和方法。

“理解运用计算生态，培养集成创新思维”是我们期望传达的教学理念。本书以此为出发点，试图实现两个目标：使读者掌握一门终身受用的编程语言（Python 语言）；使读者体验运用计算生态解决实际问题的过程和思路。期待读者能通过 Python 语言的学习，真正走进计算世界，享受创新的乐趣！

本书成稿过程一波三折，先后历时一年，随着教学理念的不断发展完善以及教学经验的积累，本书大部分内容被推翻或重写过多次，本书超过 90%的实例都是作者原创。此外，借助作者在中国大学 MOOC 平台（www.icourses.cn/imooc）上开设的“Python 语言程序设计”课程及“Python 系列专题”课程，读者可愉快地体验在线学习的乐趣。

为了辅助教师开展教学和配合读者学习，本书提供 Python 语言相关的资源平台，其中本

书所配套电子资源均上传至Python123主平台（www.python123.org）。希望扩展练习的读者或计划组织课内教学的老师，也可以使用Python123题库和评测系统（www.python123.io）。

本书作者均来自于北京理工大学，学校发展教育教学的决心和行动给予作者莫大的支持，在此表示感谢。此外，还要感谢在本书撰写和出版过程中给予过帮助的人，包括研究生/本科生袁炜佳、李天龙、杨雅婷、刘苗苗、魏煜等。本书得到了教育部谷歌产学合作项目、北京市教育委员会“北京高等学校青年英才计划项目”、北京理工大学优秀青年教师资助计划项目（教学提升）和北京理工大学校级十三五规划教材项目的资助，在此一并表示感谢。限于水平，不足之处在所难免，敬请读者和同行批评指正。作者的电子邮件地址是songtian@bit.edu.cn。

编　者

2016年12月

索　　引

全书实例索引

表 1　全书 21 个实例索引

实例编号	实例名称	对应章节
实例 1	温度转换	2.1
实例 2	Python 蟒蛇绘制	2.3
实例 3	天天向上的力量	3.4
实例 4	文本进度条	3.7
实例 5	身体质量指数 BMI	4.3
实例 6	π 的计算	4.6
实例 7	七段数码管绘制	5.4
实例 8	科赫曲线绘制	5.7
实例 9	基本统计值计算	6.3
实例 10	文本词频统计	6.6
实例 11	Python 之禅	6.7
实例 12	图像的字符画绘制	7.3
实例 13	CSV 格式的 HTML 展示	7.5
实例 14	CSV 和 JSON 格式相互转换	7.8
实例 15	体育竞技分析	8.2
实例 16	pip 安装脚本	8.7
实例 17	图像的手绘效果	9.3
实例 18	科学坐标图绘制	9.5
实例 19	多级雷达图绘制	9.6
实例 20	中国大学排名爬虫	10.4
实例 21	搜索关键词自动提交	10.5

全书函数库索引

表 2　全书 11 个函数库索引

函数库编号	函数库名称	对应章节
模块 1	math	3.3

续表

函数库编号	函数库名称	对应章节
模块 2	random	4.5
模块 3	datetime	5.3
模块 4	jieba	6.5
模块 5	PIL	7.2
模块 6	json	7.7
模块 7	pyinstaller	8.4
模块 8	numpy	9.2
模块 9	matplotlib	9.4
模块 10	requests	10.2
模块 11	beautifulsoup4	10.3

全书拓展概念索引

表 3 全书 65 个拓展概念索引

章节	对应小节	拓展概念
第 1 章	1.1	摩尔定律
	1.2	通用编程语言
	1.3	开源软件
	1.4	斐波那契数列
	1.5	人工智能和图灵测试
	1.6	向后兼容
第 2 章	2.1	算法
	2.2	保留字
	2.3	面向对象编程
	2.4	RGB 颜色
第 3 章	3.1	高精度浮点运算类型
	3.2	模运算
	3.3	伽玛函数
	3.4	GRIT：成功的关键
	3.5	特殊的格式化控制字符
	3.6	字符串和字节流
	3.7	进度条设计方法
第 4 章	4.1	程序的描述方式
	4.2	通往天堂的选择
	4.3	中国居民膳食指南
	4.4	科幻电影中的循环故事
	4.5	伪随机数和真随机数
	4.6	圆周率日
	4.7	异常和错误

续表

章　节	对应小节	拓展概念
第 5 章	5.1	函数式编程
	5.2	指针和引用
	5.3	1970 年 1 月 1 日
	5.4	计算机的硬件时钟
	5.5	紧耦合和松耦合
	5.6	数学归纳法
	5.7	分形几何
	5.8	Python 使用手册
第 6 章	6.1	哈希运算
	6.2	列表和数组
	6.3	中位数的含义
	6.4	索引
	6.5	第三方库
	6.6	自然语言处理
	6.7	代码的艺术
第 7 章	7.1	文件的换行符
	7.2	CMYK 色彩
	7.3	位图和矢量图
	7.4	Python 的 csv 标准库
	7.5	Web 前端开发
	7.6	XML
	7.7	序列化
	7.8	Python 的数据类型转换
第 8 章	8.1	ENIAC
	8.2	模拟和仿真
	8.3	软件开发模型
	8.4	动态链接
	8.5	Python——构建计算生态的编程语言
	8.6	whl 格式
	8.7	PyPI 的权重值
第 9 章	9.1	离散和连续
	9.2	运算规则
	9.3	灰度值
	9.4	字体
	9.5	科学计算可视化
	9.6	兴趣是最好的老师
第 10 章	10.1	Robots 排除协议
	10.2	HTTP 的 GET 和 POST
	10.3	正则表达式
	10.4	大学排名
	10.5	CAPTCHA 验证码

全书表格索引

表 4　全书 76 个表格索引

章　　节	对应小节	表格名称
第 2 章	2.2	表 2.1　Python 3 的 33 个保留字列表
	2.4	表 2.2　部分典型 RGB 颜色对照表
第 3 章	3.1	表 3.1　整数类型的 4 种进制表示
	3.2	表 3.2　内置的数值运算操作符（共 9 个）
	3.2	表 3.3　内置的数值运算函数（共 6 个）
	3.2	表 3.4　内置的数字类型转换函数（共 3 个）
	3.3	表 3.5　math 库的数学常数（共 4 个）
	3.3	表 3.6　math 库的数值表示函数（共 16 个）
	3.3	表 3.7　math 库的幂对数函数（共 8 个）
	3.3	表 3.8　math 库的三角运算函数（共 16 个）
	3.3	表 3.9　math 库的高等特殊函数（共 4 个）
	3.5	表 3.10　基本的字符串操作符（共 5 个）
	3.5	表 3.11　内置的字符串处理函数（共 6 个）
	3.5	表 3.12　常用的内置字符串处理方法（共 16 个）
	3.7	表 3.13　进度条设计函数
第 4 章	4.2	表 4.1　Python 的关系操作符（共 6 个）
	4.3	表 4.2　BMI 指标分类
	4.5	表 4.3　random 库的常用函数（共 9 个）
	4.6	表 4.4　不同抛点数产生的精度和运行时间
第 5 章	5.3	表 5.1　datetime 类的常用属性（共 9 个）
	5.3	表 5.2　datetime 类常用的时间格式化方法（共 3 个）
	5.3	表 5.3　strftime()方法的格式化控制符
	5.8	表 5.4　Python 的内置函数列表（共 68 个）
第 6 章	6.1	表 6.1　序列类型的通用操作符和函数（共 12 个）
	6.1	表 6.2　集合类型的操作符（共 10 个）
	6.1	表 6.3　集合类型的操作函数或方法（共 10 个）
	6.2	表 6.4　列表类型特有的函数或方法（14 个）
	6.4	表 6.5　字典类型的函数和方法
	6.5	表 6.6　jieba 库常用分词函数（7 个）
第 7 章	7.1	表 7.1　文件的打开模式（共 7 个）
	7.1	表 7.2　文件内容读取方法（共 4 个）
	7.1	表 7.3　文件内容写入方法（共 3 个）
	7.2	表 7.4　Image 类的图像读取和创建方法（共 5 个）
	7.2	表 7.5　Image 类的常用属性（共 4 个）
	7.2	表 7.6　Image 类的序列图像操作方法（共 2 个）
	7.2	表 7.7　Image 类的图像转换和保存方法（共 3 个）
	7.2	表 7.8　Image 类的图像旋转和缩放方法（共 2 个）

续表

目　录

第一部分　初识 Python 语言

第二部分　深入 Python 语言

第三部分 运用 Python 语言

第一部分　初识 Python 语言

Mooc 课程：
中国大学 MOOC
"Python 语言程序设计"课程



本部分主要讲解初识 Python 语言的那些事儿，让读者能够快速入门，跨越那段最朦胧、最期待的未知地带，了解 Python 语言的基本概念并建立对程序设计方法的基本理解。这一部分的学习目标是编写 10 行左右的 Python 程序。

本部分包括 3 章内容（第 1、2 章和附录 A），分别如下：

第 1 章　程序设计基本方法

第 2 章　Python 程序实例解析

附录 A　极简计算机基础

第 1 章主要面向初学编程语言的读者，重点讲解编写程序最基本的 IPO 方法，介绍 Python 语言安装和运行过程，说明 Python 语言的版本更迭和选择。

第 2 章讲解两个 Python 程序实例，围绕实例介绍 Python 语言的语法元素和编程模式，帮助读者建立 Python 语言编程的总体概念。

附录 A 补充介绍编写程序所需要了解的计算机基础概念，从数据到万维网，从存储程序结构到虚拟化，采用极简叙述方式撰写，适合读者在学习本书内容时复习参考。

对于初学程序设计的读者，除附录 A 作为参考外，建议按照章节顺序逐步学习。对于有一定程序设计基础但不熟悉 Python 语言的读者，建议了解 Python 语言安装环境后重点学习第 2 章内容。由于附录 A 采用极简叙述方式撰写，可以作为读者的阅读资料。

第 1 章　程序设计基本方法

电子教案 1-1：程序设计基本方法



在这个国家，每个人都应该学习如何编程，因为它教你如何思考。

Everybody in this country should learn how to program a computer, because it teaches you how to think.

——史蒂夫·乔布斯（Steve Jobs）

苹果公司创始人

学习目标

（1）理解硬件和软件在计算机系统中的不同作用。

（2）了解程序设计语言的发展过程。

（3）理解 Python 语言的特点以及其重要性。

（4）掌握 Python 语言 Hello 程序的编写方法。

（5）掌握 Python 语言开发和运行环境的配置方法。

（6）理解编写程序的 IPO 方法。

（7）了解 Python 版本更迭过程和新旧版本的主要区别。

Hello World 是 1978 年 Brian Kernighan 经典著作《C 程序设计语言》的第一个例子。这段简短的代码逐渐演变成了具有特殊象征意义的里程碑。时至今日，这个程序几乎是每门编程语言中不可替代的首个程序。接下来，读者将看到如何用 Python 语言来编写简洁优美且跨平台的 Hello World 程序。

向计算机世界发出你的问候吧！

1.1 计算机的概念

要点： 计算机是根据指令操作数据的设备，具备功能性和可编程性两个基本特性。

计算机，不可否认，是人类最伟大的发明之一。

“Computer”，最初指专门负责计算的人，到了 20 世纪中期逐渐演变为计算设备，当代特指计算机。

计算机的故事要从人类始于久远但延续至今的计算需求说起。人类为何需要计算？很显然，人类在敬畏自然、认识自然甚至试图驾驭自然的过程中，为了认识自然现象、分析自然规律，需要进行量化计算；人类社会对有限资源的分配、对人类活动的有效管理，需要进行优化计算；人类探索思维空间的数学、逻辑和哲学问题，需要进行推理演算。到了 21 世纪，人类间便捷和高效的通信需求推动了网络计算的发展，计算需求已经深入到人类的日常生活中，无处不在。

求解计算问题的方法由计算科学来研究，具体的计算任务由计算设备来完成。广义上讲，计算设备包含但不限计算机。

计算机的定义有很多种，如下定义更符合计算机的本质：计算机是根据指令操作数据的设备（A computer is a machine that manipulates data according to a list of instructions）。从定义可以看出，计算机有两个基本特性：功能性和可编程性。功能性指对数据的操作，表现为数据计算、输入输出处理和结果存储等。可编程性指它可以根据一系列指令自动地、可预测地、准确地完成操作者的意图。

理解计算机应该结合计算机的两个特性。只要设备具备了计算的功能性和操作的可编程性，就可以看作是计算机。判断一个计算设备是否属于计算机并不依靠其制造材质，计算机不一定是电子的。例如，计算机前沿领域的光计算机、量子计算机、超导计算机、生物计算机等新形态计算机都不是建立在电子学基础上，但它们都表达了计算机的概念，也属于计算机类别。除特殊说明外，本书后续内容中的“计算机”均指电子计算机。

计算机技术发展主要围绕计算机的功能性和可编程性展开。一方面，计算机硬件所依赖的集成电路规模按照摩尔定律以指数方式增长，计算机运行速度也接近几何级数快速增加，计算机所能高效支撑的功能不断丰富发展。另一方面，表达计算机可编程性的程序设计语言也在经历从机器语言、汇编语言到高级语言的发展过程，并逐步朝着更接近自然语言的方向发展。

拓展： 摩尔定律

摩尔定律（Moore’s Law）是计算机发展历史上最重要的预测法则，注意，它不是物理或自然法则，它由英特尔（Intel）公司创始人之一戈登·摩尔（Gorden E.

Moore）于 1965 年提出的。摩尔定律指出，单位面积集成电路上可容纳晶体管的数量约每两年翻一倍。由于计算机中几乎所有的重要部件，例如，CPU、内存、硬盘、网络接口等，都由集成电路实现，摩尔定律实际上揭示了 1965 年至今仍在高速发展的半导体技术趋势，进而，摩尔定律成为计算机性能水平的一个重要预测法则。

如果穿越时间回到 1965 年，计算机还只是科学研究装置，不仅如此，当时计算器尚未诞生（第一个便携式电子计算器在 1970 年才诞生）。摩尔定律所预测的晶体管数量及所表达的计算机性能指数发展趋势令人十分震撼。技术发展带来的类似震撼至今仍然“默默且显著地”改变着人类的生活，如同 10 年前大多数人无法想象手机会运行几百个不同应用一样。换个角度思考，如果能够利用技术视角和专业精神去审视那些当代重要的预测法则，将使这些技术人才能够“借助规律预测未来”，更好地迎接每一个技术震撼带来的时代变革，成为时代创新的弄潮儿。

自 1946 年第一台数字电子计算机诞生以来，计算机技术先后经历了几次重大技术发展变革，具有鲜明的时代性，与之相适应，计算机在功能性和可编程性两方面的体现也不相同。本书将这种计算机技术发展的时代性总结为 4 个阶段。

第一阶段：1946—1981 年，“计算机系统结构阶段”。这个阶段始于 1946 年，以全球首台数字计算机 ENIAC 诞生为标志。在这个阶段，计算机技术主要围绕计算机系统结构设计开展，服务于科学计算和商业数值类计算，产生了超级计算机、高性能计算机、工作站、个人计算机等不同类型的计算机系统。与这个时期计算机有限的计算性能和功能相对应，计算机的可编程性主要表现为合理划分软/硬件接口、控制计算部件完成高速运算，程序设计需要在程序逻辑和系统结构之间、处理能力和存储容量之间、计算和通信之间寻找优化和折中。这个阶段的计算需求催生了执行高效的 C 语言（1972 年）。C 语言的可编程性体现在通过指针优化底层内存的使用，进而使程序在有限计算资源下高速运行。计算机技术的第一个阶段持续了 35 年，随着以 IBM PC 为代表的个人计算机的诞生（1981 年），计算机技术进入了面向大众的新阶段。

第二阶段：1982—2007 年，“计算机网络和视窗阶段”。这个阶段始于 1982 年，以面向全球子网间组网的 TCP/IP 网络协议的标准化为标志，互联网（Internet，最初含义是连接子网的网络）时代到来了。在这个阶段，计算机技术主要围绕网络技术、视窗技术、多媒体技术发展，以个人计算机和服务器为主要计算平台，计算机技术提供满足个人计算需求的视窗应用和网络服务。由于网络将不同类型系统互联互通，在多种操作系统上执行同一个程序的跨平台特性成为计算机编程的迫切需求，由此诞生了具备跨平台功能的 Java 语言（1995 年）。与此同时，由于微软 Windows 操作系统在个人计算机领域的高度普及，视窗应用“所见即所得”的开发需求催生了 Visual C++（VC）、Visual Basic（VB）（1991 年）等视窗编程语言。计算机技术的第二个阶段持续了 25 年，随着美国苹果公司 iPhone 智能手机的推出（2007 年）和广泛普及，计算机技术进入了面向移动网络应用的新阶段。

第三阶段：2008 年至今，“复杂信息系统阶段”。这个阶段始于 2008 年，以安卓（Andriod）开源移动操作系统的发布为起点，一批新的计算概念和技术几乎同时提出并显著推动了计算技术的升级换代，这些概念包括移动互联网、多核众核、云计算、可信计算、大数据、可穿戴计算、物联网、互联网+等。这些概念的提出反映了计算平台和应用的多样性，也带来了更复杂的安全问题。虽然概念很多，但没有以哪个概念为主引领技术发展，这说明，计算机技术的发展已经进入了复杂信息系统阶段，这个阶段很难有任何一个技术领域独领风骚，任何系统都需要不间断地完善才能够提供更加安全可靠及更佳用户体验的功能，系统之间通过网络、开源项目和社交关系等高度关联，人类将会逐渐认识到计算机系统的复杂性会到达人类所能掌控的边界。面对复杂的功能性和紧迫的迭代周期，计算机需要更高抽象级别的程序设计语言来表达可编程性，Python 语言（2008 年 3.0 版本）已经成为这个阶段计算机系统的主流编程语言。

第四阶段：约 20 年后某个时期开始，“人工智能阶段”。随着深度学习、开源硬件、智能机器人、在线搜索引擎、量子计算等技术的发展，未来某个时期将会出现人工智能主导计算的技术阶段，计算机技术将结合智能技术展示更加友好的交互方式和用户体验。此时，计算机或许已经没有了独立的载体，它将通过网络、数据和机器整合一切可用自然资源，逐步接管人类所有非创造性工作，计算机技术将进入一个未知的新阶段。

纵观计算机技术短暂的发展历史，计算机功能性和可编程性的发展存在相互促进的关系。一方面，计算机的编程方式逐步让计算机更有效地理解人类意图，进而丰富计算机的功能。另一方面，计算机的功能发展进一步促进了编程方式的发展。应该说，计算机正在借助人类智慧不断“进化”。

思考与练习

1.1 计算机的定义是什么？它有哪两个显著特点？

1.2 请调研并阐述不少于 3 个计算机领域中类似摩尔定律的预测法则或评估法则。

1.3 请列出并阐述不少于 5 个近 10 年出现的计算机技术名词。

1.2 程序设计语言

要点： 程序设计语言的执行方式包括编译执行和解释执行两种。

1.2.1 程序设计语言概述

程序设计语言是计算机能够理解和识别用户操作意图的一种交互体系，它按照

特定规则组织计算机指令，使计算机能够自动进行各种运算处理。按照程序设计语言规则组织起来的一组计算机指令称为计算机程序。程序设计语言也叫编程语言，本书两种说法都会出现并交替使用。

程序设计语言包括 3 个大类：机器语言、汇编语言和高级语言。

机器语言是一种二进制语言，它直接使用二进制代码表达指令，是计算机硬件可以直接识别和执行的程序设计语言。例如，执行数字 2 和 3 的加法，16 位计算机上的机器指令为：11010010 00111011，不同计算机结构的机器指令不同。

直接使用机器语言编写程序十分繁冗，同时，二进制代码编写的程序难以阅读和修改，因此，汇编语言诞生了，它使用助记符与机器语言中的指令进行一一对应，在计算机发展早期能帮助程序员提高编程效率。例如，执行数字 2 和 3 的加法，汇编语言指令为：add 2, 3, result，运算结果写入 result。与机器语言类似，不同计算机结构的汇编指令不同。由于机器语言和汇编语言都直接操作计算机硬件并基于此设计，所以它们统称为低级语言。

高级语言与低级语言的区别在于，高级语言是接近自然语言的一种计算机程序设计语言，可以更容易地描述计算问题并利用计算机解决计算问题。例如，执行数字 2 和 3 加法，高级语言代码为：result = 2 + 3，这个代码只与编程语言有关，与计算机结构无关，同一种编程语言在不同计算机上的表达方式是一致的。

如果能像科幻电影中的情节一样，用人类语言驱动计算机将是最完美的事情。遗憾的是，尽管许多一流科学家为此做过很多努力，仍然无法在可预见的未来设计出能完全理解人类语言的计算机。

诚然，即使计算机能理解人类语言，人类语言也不适合描述复杂算法，这是因为人类语言具有不严密和模糊的缺点。例如，“我看见一个人在公园，带着望远镜。”这句话，基于常识和经验，交谈双方大多数情况下能够理解彼此表达的意思，但深究一下，究竟是“我”带着望远镜，还是“一个人”带着望远镜呢？这种模糊性也经常产生错误理解和歧义。相比人类语言，程序设计语言的结构在语法上十分精密，在语义上定义准确。

第一个广泛应用的高级语言是诞生于 1972 年的 C 语言。随后 40 多年来先后诞生了 600 多种程序设计语言，但是大多数语言由于应用领域的狭窄退出了历史舞台。下面列出的是至今还经常使用的程序设计语言，包括 C、C++、C#、Go、HTML、Java、JavaScript、PHP、Python、SQL、Verilog 等。一般来说，通用编程语言比专用于某些领域的编程语言生命力更强。

拓展： 通用编程语言

通用编程语言指能够用于编写多种用途程序的编程语言（相对于专用编程语言）。例如，Python 语言是一个通用编程语言，可以用于编写各种类型的应用，该语言的语法中没有专门用于特定应用的程序元素。HTML 语言则是一个专用编程语言，它利用超链接将文本、图像、音/视频等资源组织起来形成 Web 页面。尽管有些编程语言不包含针对特定应用的程序元素，但由于语言所应用的领域比较狭窄，也被认为是专用编程语言。

常用编程语言中，C、C++、C#、Go、Java、Python 是通用编程语言，HTML（Web 页面超链接语言）、JavaScript（Web 浏览器端动态脚本语言）、MATLAB（基于矩阵运算的科学计算语言）、PHP（Web 服务器端动态脚本语言）、SQL（数据库操作语言）、Verilog（硬件描述语言）是专用编程语言。

1.2.2 编译和解释

高级语言按照计算机执行方式的不同可分成两类：静态语言和脚本语言。这里所说的执行方式是指计算机执行一个程序的过程，静态语言采用编译执行，脚本语言采用解释执行。无论哪种执行方式，用户的使用方法可以是一致的，如通过鼠标双击执行一个程序。

编译是将源代码转换成目标代码的过程，通常，源代码是高级语言代码，目标代码是机器语言代码，执行编译的计算机程序称为编译器（Compiler）。如图 1.1 展示了程序的编译过程，其中，虚线表示目标代码被计算机运行。编译器将源代码转换成目标代码，计算机可以立即或稍后运行这个目标代码。

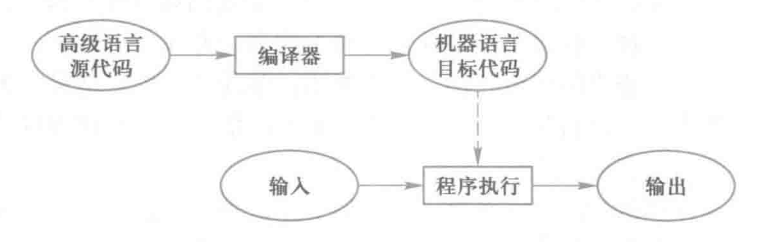


图 1.1 程序的编译和执行过程

解释是将源代码逐条转换成目标代码同时逐条运行目标代码的过程。执行解释的计算机程序称为解释器（Interpreter）。如图 1.2 展示了程序的解释过程。其中，高级语言源代码与数据一同输入给解释器，然后输出运行结果。

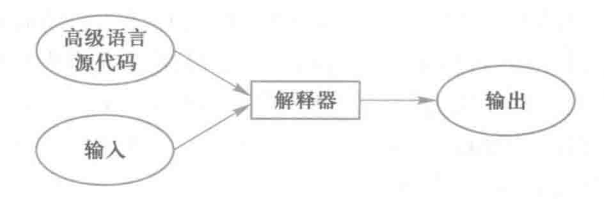


图 1.2 程序的解释和执行过程

解释和编译的区别在于编译是一次性地翻译，一旦程序被编译，不再需要编译程序或者源代码。解释则在每次程序运行时都需要解释器和源代码。这两者的区别类似于外语资料的翻译和实时的同声传译。

编译过程只进行一次，所以，编译过程的速度并不是关键，目标代码的运行速度是关键。因此，编译器一般都集成尽可能多的优化技术，使生成的目标代码具备更好的执行效率。然而，解释器却不能集成太多优化技术，因为代码优化技术会消

耗运行时间，使整个程序的执行速度受到影响。

采用编译方式有如下好处。

（1）对于相同源代码，编译所产生的目标代码执行速度更快。

（2）目标代码不需要编译器就可以运行，在同类型操作系统上使用灵活。

采用解释方式有如下好处。

（1）解释执行需要保留源代码，程序纠错和维护十分方便。

（2）只要存在解释器，源代码可以在任何操作系统上运行，可移植性好。

简单地说，解释执行、逐条运行用户编写的代码，没有纵览全部代码的性能优化过程，因此执行性能略低，但可以支持跨硬件或操作系统平台，保留源代码对升级维护十分有利，适合非性能关键的程序运行场景。

采用编译执行的编程语言是静态语言，如C语言、Java语言；采用解释执行的编程语言是脚本语言，如JavaScript语言、PHP语言。Python语言是一种被广泛使用的高级通用脚本编程语言，虽采用解释执行方式，但它的解释器也保留了编译器的部分功能，随着程序运行，解释器也会生成一个完整的目标代码。这种将解释器和编译器结合的新解释器是现代脚本语言为了提升计算性能的一种有益演进。

1.2.3　计算机编程

——为什么要学习计算机编程？

——因为“编程是件很有趣的事儿”！

编程能够训练思维。编程体现了一种抽象交互关系、形式化方法执行的思维模式，称为“计算思维”。计算思维是区别于以数学为代表的逻辑思维和以物理为代表的实证思维的第三种思维模式。编程是一个求解问题的过程，首先需要分析问题，抽象内容之间的交互关系，设计利用计算机求解问题的确定性方法，进而通过编写和调试代码解决问题，这是从抽象问题到解决问题的完整过程。计算思维的训练过程能够促进人类思考，增进观察力和深化对交互关系的理解。

编程能够增进认识。编写程序不单纯是求解计算题，它要求作者不仅要思考解决问题的方法，更要思考如何让程序有更好的用户体验、更高的执行效率和更有趣的展示效果。不同群体、不同时代、不同文化对程序使用有着不同理解，编程需要对时代大环境和使用群体小环境有更多认识，从细微处给出更好的程序体验，这些思考和实践将帮助程序作者加深对用户行为以及社会和文化的认识。

编程能够带来乐趣。利用一台计算机，编程能够提供展示自身思想和能力的舞台，将所思所想变为现实。编程的开始有各种动机，或者去展示自己的青春风采，或者讽刺不文明的社会现象，或者向爱慕的对象表达情愫，所有这些想法都可以通过编写程序变成现实，并通过互联网零成本分发获得更大的影响力。这些努力会让世界增加新的颜色、让自己变得更酷、提升心理满足感和安全感。

编程能够提高效率。计算机已经成为当今社会的普通工具，掌握一定编程技术有助于更好地利用计算机解决所面对的计算问题。例如，对于个人照片，可以通过程序读取照片元属性自动进行归类整理。对于工作数据，可以通过程序按照特定算

法进行批处理并绘制统计图表。对于朋友圈的好文，可以通过程序实时关注随时点赞。掌握一些编程技术能够提高工作、生活和学习效率。

编程带来就业机会。程序员是信息时代最重要的工作岗位之一，国内外对程序员的缺口都在百万级及以上规模，就业前景广阔。程序员职业往往并不需要掌握多种编程语言，精通一种就能够获得就业机会。如果读者不喜欢自己的专业或现在的工作，那就认真学习程序设计，换个更有趣的工作吧！

很多读者都有一个误区，认为编程很难学。事实上，“编程不是一件很难的事儿”，因为编写程序有一定的框架和模式，只要理解了这些模式，稍加练习就会有很好的学习效果。学习一门编程语言，首先要掌握该语言的语法（既要系统掌握基本语法，又要能灵活运用）。其次要学会结合计算问题设计程序结构，从程序块、功能块角度理解并设计整个程序框架。最后要掌握解决问题的能力，即从理解计算问题开始，设计问题的解决方法，并通过编程语言来实现。学习计算机编程的重点在于练习。不仅要多看代码，照着编写，调试运行，还要在参考代码思路基础上独立编写，学会举一反三。为了帮助读者掌握程序设计能力，本书将大量编程概念和语法通过有趣的实例组织起来，并展示 Python 语言的魅力和力量。希望这样的设计能够为读者的 Python 学习过程带来快乐和价值。

思考与练习

1.4 CPU 可以直接理解什么类型的程序设计语言？

1.5 请阐述编译和解释两种执行方式的区别和各自的优缺点。

1.6 结合读者的实际情况，请列出不少于 3 个学习编程语言的理由。

1.3 Python 语言概述

要点： Python 语言是一个语法简洁、跨平台、可扩展的开源通用脚本语言。

1.3.1 Python 语言的发展

Python 语言诞生于 1990 年，由 Guido van Rossum 设计并领导开发。1989 年 12 月，Guido 考虑启动一个开发项目以打发圣诞节前后的时间，所以决定为当时正在构思的一个新的脚本语言写一个解释器，因此在次年诞生了 Python 语言。该语言以“Python”命名源于 Guido 对当时一部英剧“Monty Python’s Flying Circus”的极大兴趣。也许 Python 语言的诞生是个偶然事件，但 20 多年持续不断的发展将这个偶然事件变成了计算机技术发展过程中的一件大事。

Python 语言是开源项目的优秀代表，其解释器的全部代码都是开源的，可以在

Python 语言的主网站（https://www.python.org/）自由下载。Python 软件基金会（Python Software Foundation，PSF）作为一个非营利组织，拥有 Python 2.1 版本之后所有版本的版权，该组织致力于更好推进并保护 Python 语言的开放性。

2000 年 10 月，Python 2.0 正式发布，标志着 Python 语言完成了自身涅槃，解决了其解释器和运行环境中的诸多问题，开启了 Python 广泛应用的新时代。2010 年，Python 2.x 系列发布了最后一版，其主版本号为 2.7，用于终结 2.x 系列版本的发展，并且不再进行重大改进。

2008 年 12 月，Python 3.0 正式发布，这个版本在语法层面和解释器内部做了很多重大改进，解释器内部采用完全面向对象的方式实现。这些重要修改所付出的代价是 3.x 系列版本代码无法向下兼容 Python 2.0 系列的既有语法，因此，所有基于 Python 2.0 系列版本编写的库函数都必须修改后才能被 Python 3.0 系列解释器运行。

Python 语言经历了一个痛苦但令人期待的版本更迭过程，从 2008 年开始，用 Python 编写的几万个函数库开始了版本升级过程，至今，绝大部分 Python 函数库和 Python 程序员都采用 Python 3.0 系列语法和解释器。本书 1.6 节将具体阐述 Python 2.x 和 Python 3.x 的不同。

“Python 2.x 已经是遗产，Python 3.x 是这个语言的现在和未来”。

拓展： 开源软件

开源软件（Open-Source Software）是一类开放源代码软件的统称，这类软件的源代码在特定许可协议范围内，可以被任何人学习、修改甚至发布，开源软件更多定义和资源请参考开源软件社区网站（https://opensource.org）。开源软件从 1998 年开始被众多资深程序员定义并推动，至今，世界上有几十万个开源软件项目，覆盖几乎所有软件应用领域。开源软件为计算机技术快速发展扫清了知识产权障碍，降低了学习成本，互联网也进一步推动了开源软件的传播。如今，全球 80%以上的服务器运行开源软件，每年为用户节省 600 亿美元以上的使用成本。开源软件成就了当代计算机技术的发展程度，在深层次上影响着未来信息技术的发展速度和普及程度。

1.3.2　编写 Hello 程序

学习编程语言有一个惯例，即运行最简单的 Hello 程序，该程序功能是在屏幕上打印输出“Hello World”。这个程序虽小，但却是初学者接触编程语言的第一步。使用 Python 语言编写的 Hello 程序只有一行代码，如下：

```
print("Hello World")
```

上述代码中，**print()**表示将括号中引号内的信息输出到屏幕上。该代码在 Python 运行环境（1.4 节将具体介绍 Python 运行环境的配置）中的执行效果如下：

```
>>>print("Hello World")
Hello World
```

其中，第一行的“>>>”是 Python 语言运行环境的提示符，其表示可以在此符号后面输入 Python 语句。第二行是 Python 语句的执行结果。

Python 语言的 Hello 程序似乎与人类语言类似，即通过一行语句就完成了输出一段文本的任务。其他编程语言的 Hello 程序并不这样简洁，如下是 C 语言的 Hello 程序，对比一下。

```
#include <stdio.h>
int main(void)
{
    printf("Hello World\n");
    return 0;
}
```

C 语言程序除了向屏幕输出“Hello World”的 printf 语句外，还包含了 include、int、main、printf、return 等其他辅助元素，这里就不具体介绍了。这个最小的例子只是一个缩影，Python 的简洁性在编程语言领域是公认的。同样功能的程序，Python 语言实现的代码行数仅相当于 C 语言的 1/10～1/5(其简洁程度取决于程序的复杂度和规模)。更少的代码行数、更简洁的表达方式将带来更少的程序错误、更快的程序开发速度和更好的可读性。

1.3.3 Python 语言的特点

Python 语言是一种被广泛使用的高级通用脚本编程语言，具有很多区别于其他语言的特点，这里仅列出如下一些重要特点。

（1）语法简洁：实现相同功能，Python 语言的代码行数仅相当于其他语言的 1/10～1/5。

（2）与平台无关：作为脚本语言，Python 程序可以在任何安装解释器的计算机环境中执行，因此，用该语言编写的程序可以不经修改地实现跨平台运行。

（3）粘性扩展：Python 语言具有优异的扩展性，体现在它可以集成 C、C++、Java 等语言编写的代码，通过接口和函数库等方式将它们“粘起来”(整合在一起)。此外，Python 语言本身提供了良好的语法和执行扩展接口，能够整合各类程序代码。

（4）开源理念：对于高级程序员，Python 语言开源的解释器和函数库具有强大的吸引力，更重要地，Python 语言倡导的开源软件理念为该语言发展奠定了坚实的群众基础。

（5）通用灵活：Python 语言是一个通用编程语言，可用于编写各领域的应用程序，这为该语法提供了广阔的应用空间。几乎各类应用，从科学计算、数据处理到

人工智能、机器人，Python 语言都能够发挥重要作用。

（6）强制可读：Python 语言通过强制缩进（类似文章段落的首行空格）来体现语句间的逻辑关系，显著提高了程序的可读性，进而增加了 Python 程序的可维护性。

（7）支持中文：Python 3.0 解释器采用 UTF-8 编码表达所有字符信息。UTF-8 编码可以表达英文、中文、韩文、法文等各类语言，因此，Python 程序在处理中文时更加灵活且高效。

（8）模式多样：尽管 Python 3.0 解释器内部采用面向对象方式实现，但 Python 语法层面却同时支持面向过程和面向对象两种编程方式，这为使用者提供了灵活的编程模式。

（9）类库丰富：Python 解释器提供了几百个内置类和函数库，此外，世界各地程序员通过开源社区贡献了十几万个第三方函数库，几乎覆盖了计算机技术的各个领域，编写 Python 程序可以大量利用已有的内置或第三方代码，具备良好的编程生态。

本书后续章节将会逐步揭开 Python 语言的神秘面纱，带领读者利用 Python 语言解决一个个有趣又实用的计算问题。后续行文除个别地方外，将用“Python”代替“Python 语言”，并根据行文内容交替使用“编程”和“程序设计”。

思考与练习

1.7　请列出不少于 3 个开源软件的意义。

1.8　请列出不少于 5 个 Python 语言的特点。

1.9　在屏幕上输出“祖国，你好”的 Python 语句。

1.4　Python 语言开发环境配置

> **要点：** IDLE 是一个轻量级 Python 语言开发环境，可以支持交互式和批量式两种编程方式。

1.4.1　安装 Python 解释器

Python 语言解释器是一个轻量级的小尺寸软件，可以在 Python 语言主网站上下载（文件大小约为 25～30 MB），网址如下：

https://www.python.org/downloads/

也可以在与本书关联的中文网站上下载，网址如下：

http://www.python123.org/downloads/

其中，Python 解释器主网站下载页面如图 1.3 所示。

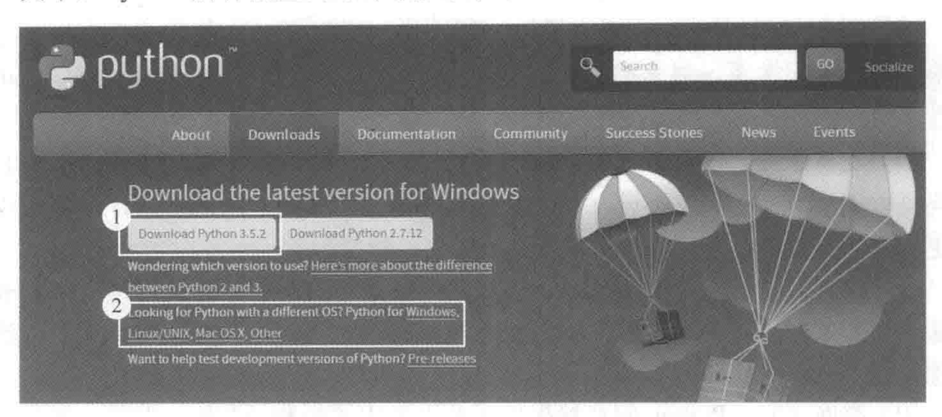


图 1.3 Python 解释器主网站下载页面

如上图，首先根据所用操作系统版本选择相应的 Python 3.x 系列安装程序。如图 1.3，单击图中矩形①内的按钮下载 Python 3.5.2 版本程序。这个位置放置的是 Python 最新的稳定版本，随着 Python 语言发展，此处会有更新的版本，本书内容统一以 3.5.2 版本为代表。以 Windows 操作系统为例，单击图中按钮下载 python-3.5.2.exe 文件。其他操作系统请选择②内相应链接，并找到对应文件进行下载。

Python 最新的 3.x 系列解释器会逐步发展，对于初学 Python 的读者，建议采用 3.5.2 或之后的版本，可以不使用最新版本。如果所在系统无法安装 3.5.2 版本，则请使用 3.4.2 版本。

双击所下载的程序安装 Python 解释器，然后将启动一个如图 1.4 所示的引导过程。在该页面中，勾选图中矩形框内的 Add Path 3.5 to PATH 复选框。

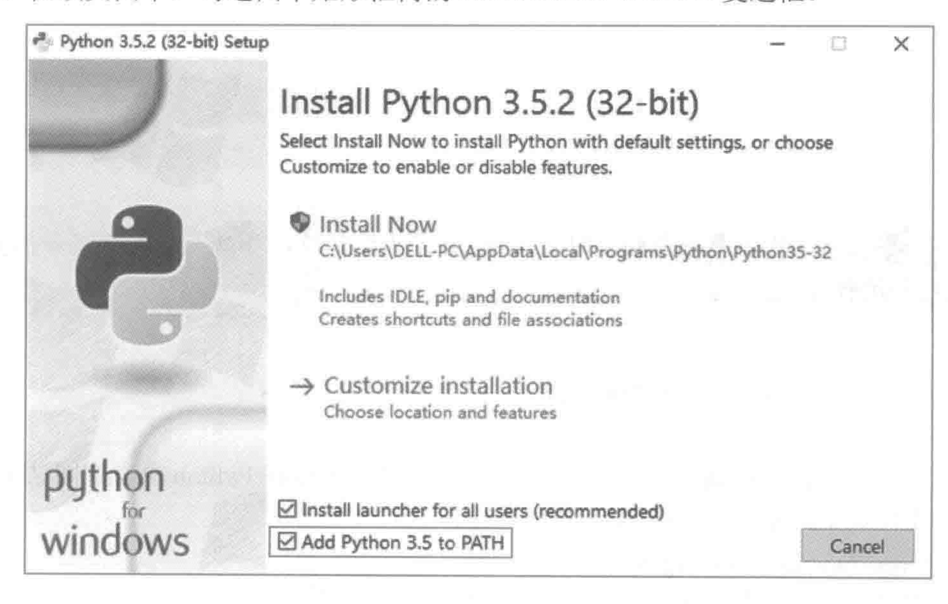


图 1.4 安装程序引导过程的启动页面

安装成功后将显示如图 1.5 所示的成功页面。

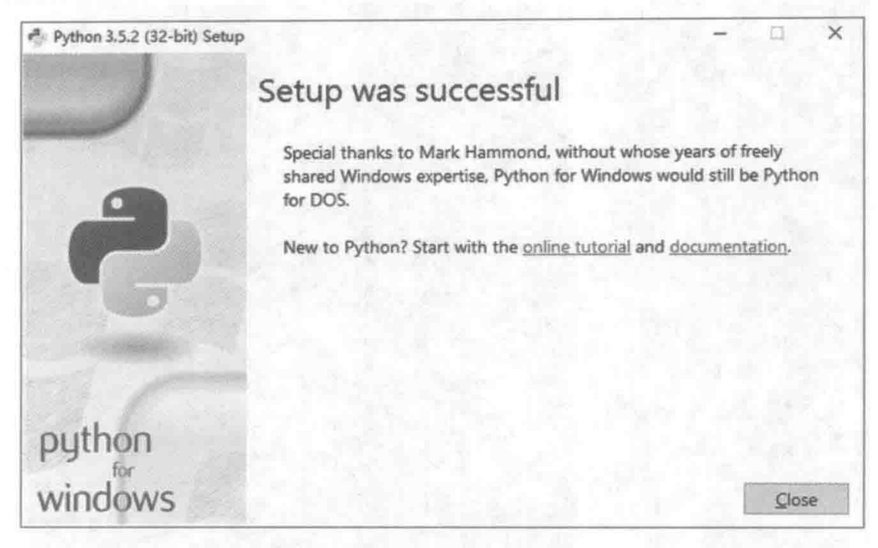


图 1.5 安装程序引导过程的成功页面

Python 安装包将在系统中安装一批与 Python 开发和运行相关的程序，其中最重要的两个是 Python 命令行和 Python 集成开发环境（Python's Integrated DeveLopment Environment，IDLE）。

1.4.2 运行 Hello 程序

运行 Python 程序有两种方式：交互式和文件式。交互式指 Python 解释器即时响应用户输入的每条代码，给出输出结果。文件式，也称为批量式，指用户将 Python 程序写在一个或多个文件中，然后启动 Python 解释器批量执行文件中的代码。交互式一般用于调试少量代码，文件式则是最常用的编程方式。其他编程语言通常只有文件式执行方式。下面以 Windows 操作系统中运行 Hello 程序为例具体说明两种方式的启动和执行方法。

1．交互式启动和运行方法

交互式有两种启动和运行方法。

第一种方法，启动 Windows 操作系统命令行工具（<Windows 系统安装目录>\system32\cmd.exe），在控制台中输入“Python”，在命令提示符>>>后输入如下程序代码：

```
print("Hello World")
```

按 Enter 键后显示输出结果“Hello World”，如图 1.6 所示。

在>>>提示符后输入 exit()或者 quit()可以退出 Python 运行环境。

第二种方法，通过调用安装的 IDLE 来启动 Python 运行环境。IDLE 是 Python 软件包自带的集成开发环境，可以在 Windows“开始”菜单中搜索关键词“IDLE”找到 IDLE 的快捷方式。如图 1.7 展示了 IDLE 环境中运行 Hello World 程序的效果。

源代码 1-1：Hello 程序

```
Command Prompt - python
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\Tian Song>python
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World")
Hello World
>>>
```

图 1.6 通过命令行启动交互式 Python 运行环境

```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:
01:18) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more
 information.
>>> print("Hello World")
Hello World
>>> |
Ln: 5 Col: 4
```

图 1.7 通过 IDLE 启动交互式 Python 运行环境

2．文件式启动和运行方法

文件式也有两种运行方法，与交互式相对应。

第一种方法，按照 Python 的语法格式编写代码，并保存为.py 形式的文件（以 Hello World 程序为例，将代码保存成文件 hello.py）。Python 代码可以在任意编辑器中编写，对于百行以内规模的代码建议使用 Python 安装包中的 IDLE 编辑器或者第

三方开源记事本增强工具 Notepad++。然后，打开 Windows 的命令行（cmd.exe），进入 hello.py 文件所在目录，运行 Python 程序文件获得输出，如图 1.8 所示。

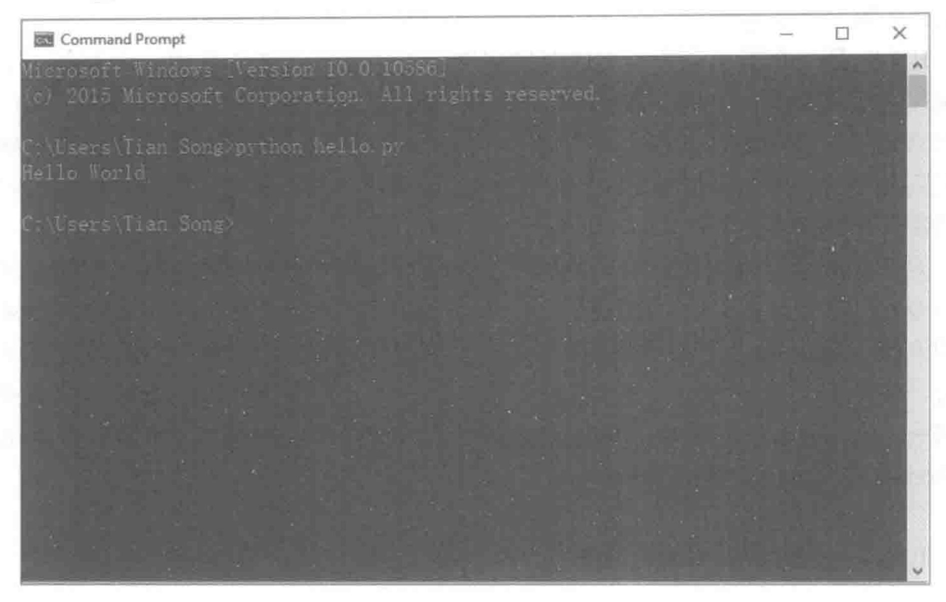


图 1.8　通过命令行方式运行 Python 程序文件

第二种方法，打开 IDLE，按快捷键 Ctrl+N 打开一个新窗口，或在菜单中选择 File→New File 选项。这个新窗口不是交互模式，它是一个具备 Python 语法高亮辅助的编辑器，可以进行代码编辑。在其中输入 Python 代码，例如，输入 Hello World 程序并保存为 hello.py 文件，如图 1.9 所示。按快捷键 F5，或在菜单中选择 Run→Run Module 选项运行该文件。

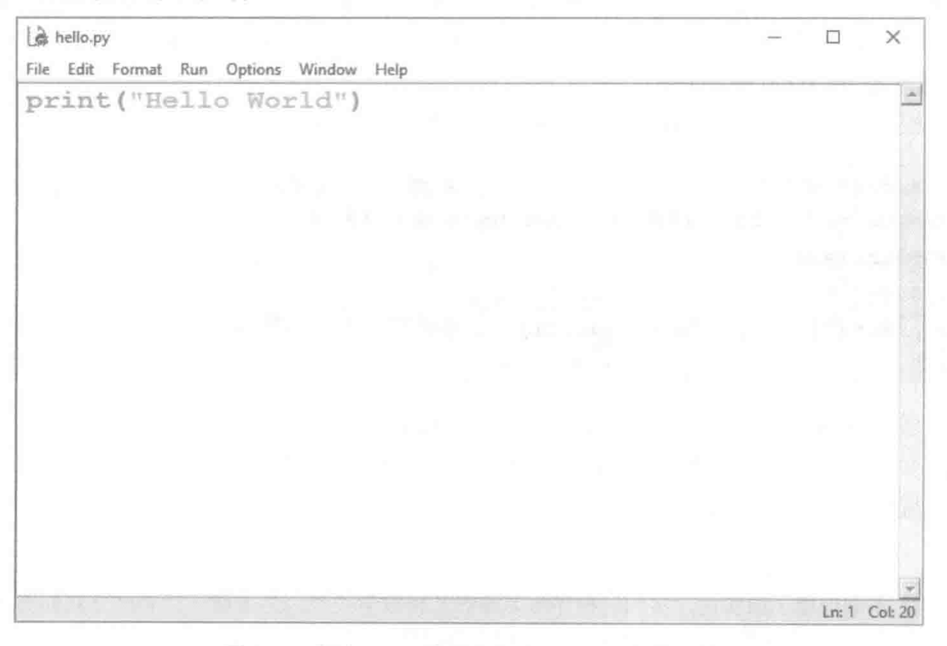


图 1.9　通过 IDLE 编写并运行 Python 程序文件

3．启动和运行方法推荐

交互式和文件式共有 4 种 Python 程序运行方法，其中，最常用且最重要的是采用 IDLE 的文件式方法。

IDLE 是一个简单有效的集成开发环境，无论交互式或文件式，它都有助于快速编写和调试代码，它是小规模 Python 软件项目的主要编写工具。本书所有程序都可以通过 IDLE 编写并运行。行文方面，对于单行代码或通过观察输出结果讲解少量代码的情况，本书采用 IDLE 交互式（由>>>开头）进行描述；对于讲解整段代码的情况，采用 IDLE 文件式。

对于完成调试确保运行无误的程序，总是希望能够通过单击鼠标直接运行（通过命令行或者 IDLE 启动都不够酷！），这个过程叫做“程序发布”，本书将在 8.4 节介绍利用 pyinstaller 库在 Windows 平台上发布程序的方法，让程序运行也酷起来！此外，附录 13.5 节介绍一个比 IDLE 更酷、更强大、更复杂的 Python 语言集成开发环境——PyCharm，它主要用于中规模及以上的软件开发项目，仅为有兴趣成为 Python 编程大牛的读者提供指引。

1.4.3 运行 Python 小程序

Hello World 程序只有一行代码，实在太小。本节给出 5 个 5 行代码左右的 Python 小程序（称为“微实例”），供读者在 IDLE 交互式和批量式两种方式下练习。这 5 个微实例分别给出了交互式执行过程和文件式内容（即全部程序内容）。

请读者暂时忽略这些实例中程序的具体语法含义，这正是接下来要学习的内容，当然，尝试理解语法也十分有益。请在 IDLE 交互环境或编辑器中编写并运行这些程序，确保它们可以输出正确结果。注意：在编辑器中输入代码时，#及后面的文字是注释，仅用来帮助读者理解程序，不影响程序执行，可以不用输入。

【微实例 1.1】圆面积的计算。

根据圆的半径计算圆的面积。交互式执行过程如下。

```
>>>radius = 25                        # 圆的半径是25
>>>area = 3.1415 * radius * radius    # 输入计算圆面积的公式
>>>print(area)
1963.4375000000002
>>>print("{:.2f}".format(area))       # 只输出两位小数
1963.44
```

微实例 1.1 的文件式内容如下，其中，首行微实例文件名对应本书的电子资源文件名[1]，读者可以将代码保存为任意名称，左侧序号为代码行号，以辅助阅读，不是程序代码的组成部分。

---

[1] 本书使用了微实例、实例、程序练习题等几种教学类型，为了便于读者区分它们对应的源代码，文件、微实例的文件名以 m 开头，程序练习题答案文件名以 a 开头，正文讲解实例的文件名以 e 开头，之后包含的是编号和功能相关的描述。

微实例 1.1　　　　m1.1CalCircleArea.py

```
1  radius = 25                              # 圆的半径是 25
2  area = 3.1415 * radius * radius          # 输入计算圆面积的公式
3  print(area)
4  print("{:.2f}".format(area))             # 只输出两位小数
```

源代码 1-2：圆面积的计算

【微实例 1.2】简单的人名对话。

对用户输入的人名给出一些不同的回应。交互式执行过程如下：

```
>>>name = input("输入姓名:")
输入姓名:郭靖
>>>print("{}同学，学好 Python，前途无量！".format(name))
郭靖同学，学好 Python，前途无量！
>>>print("{}大侠，学好 Python，大展拳脚！".format(name[0]))
郭大侠，学好 Python，大展拳脚！
>>>print("{}哥哥，学好 Python，人见人爱！".format(name[1:]))
靖哥哥，学好 Python，人见人爱！
```

微实例 1.2 的文件式内容如下：

微实例 1.2　　　　m1.2EchoName.py

```
1  name = input("输入姓名:")
2  print("{}同学，学好 Python，前途无量！".format(name))
3  print("{}大侠，学好 Python，大展拳脚！".format(name[0]))
4  print("{}哥哥，学好 Python，人见人爱！".format(name[1:]))
```

源代码 1-3：简单的人名对话



【微实例 1.3】斐波那契数列的计算。

根据斐波那契数列的定义，输出不大于 1 000 的序列元素。交互式执行过程如下。

```
>>>a, b = 0, 1
>>>while a < 1000:  # 输出不大于 1000 的序列
...     print(a, end=',')
...     a, b = b, a + b
...
0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,
```

上述代码中，连续的 3 个大于号（>>>）给出首行输入提示，3 个连续点（…）给出二级输入提示，表示延续上一行内容。这 3 个连续点在 IDLE 环境中可能出现（较早的版本）或不出现（最新版本），如果不出现则会出现连续空格。

拓展： 斐波那契数列

斐波那契数列（Fibonacci Sequence），又称为黄金分割数列，由意大利数学家 Leonardo Fibonacci 于 1202 年提出，并以其名字命名。该数列 F(*n*)定义如下：F(0)=0，F(1)=1，F(*n*)=F(*n*–2)+F(*n*–1)，其中 *n*≥2。简单说，斐波那契数列中每个数是前两个数之和。斐波那契数列中邻近两个数的比值接近黄金分割数，即 F(*n*)/F(*n*–1)接近 1.618，这个比例的极限值就是黄金分割数。此外，斐波那契数列有很多特性，所以其在搜索算法、组合数学、现代物理、化学等领域均有应用。Python 语言在表达和计算传统数学概念上十分简洁，读者可以寻找其他有趣的数学概念，并用 Python 语言计算它们。

微实例 1.3 的文件式内容如下：

源代码 1-4：斐波那契数列的计算



微实例 1.3　　　m1.3CalFibonacci.py

```
1   a, b = 0, 1
2   while a < 1000:          # 输出不大于 1000 的序列
3      print(a, end=',')
4      a, b = b, a + b
```

【微实例 1.4】同切圆的绘制。

绘制 4 个不同半径的同切圆。交互式执行过程如下：

```
>>>import turtle              # 引用 turtle 库
>>>turtle.pensize(2)          # 设置画笔宽度为 2 像素
>>>turtle.circle(10)          # 绘制半径为 10 像素的圆
>>>turtle.circle(40)          # 绘制半径为 40 像素的圆
>>>turtle.circle(80)          # 绘制半径为 80 像素的圆
>>>turtle.circle(160)         # 绘制半径为 160 像素的圆
```

随着每条语句被执行，都会启动一个窗体显示如图 1.10 中的一组同切圆。

微实例 1.4 的文件式内容如下：

源代码 1-5：同切圆的绘制

微实例 1.4　　　m1.4DrawTangentCircles.py

```
1   import turtle              # 引用 turtle 库
2   turtle.pensize(2)          # 设置画笔宽度为 2 像素
3   turtle.circle(10)          # 绘制半径为 10 像素的圆
4   turtle.circle(40)          # 绘制半径为 40 像素的圆
5   turtle.circle(80)          # 绘制半径为 80 像素的圆
6   turtle.circle(160)         # 绘制半径为 160 像素的圆
```

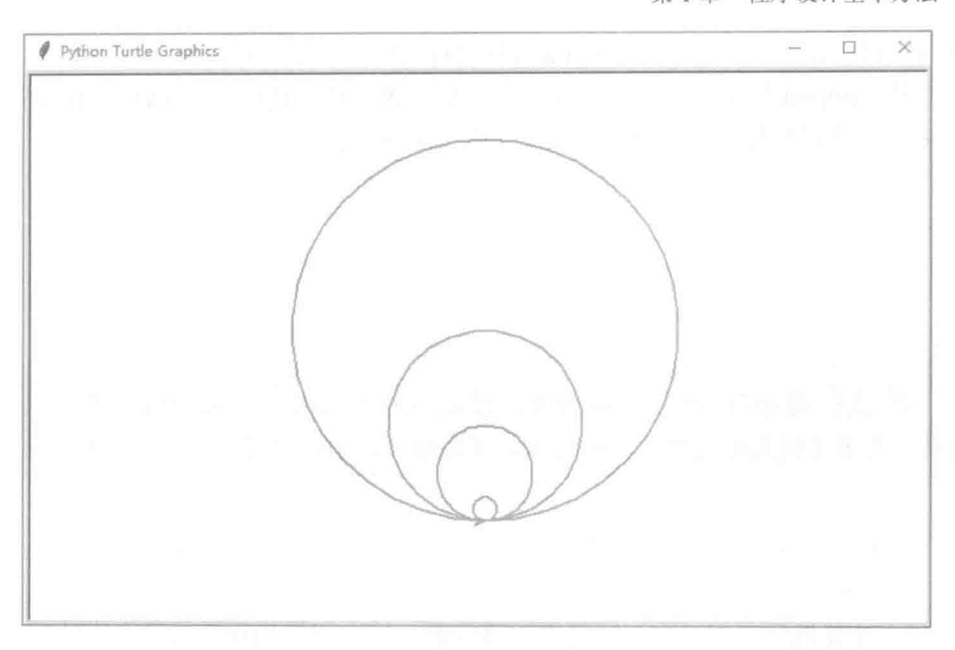


图 1.10　微实例 1.4 运行后输出的一组同切圆

【微实例 1.5】日期和时间的输出。

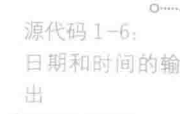
输出当前计算机的系统日期和时间。交互式执行过程如下：

```
>>>from datetime import datetime    # 引用 datetime 库
>>>now = datetime.now()             # 获得当前日期和时间信息
>>>print(now)
2016-04-04 19:09:31.505043
>>>now.strftime("%x")               # 输出其中的日期部分
2016-04-04
>>>now.strftime("%X")               # 输出其中的时间部分
19:09:31
```

微实例 1.5 的文件式内容如下：

微实例 1.5　　　m1.5PrintLocalDateTime.py

```
1  from datetime import datetime    # 引用 datetime 库
2  now = datetime.now()             # 获得当前日期和时间信息
3  print(now)
4  now.strftime("%x")               # 输出其中的日期部分
5  now.strftime("%X")               # 输出其中的时间部分
```





思考与练习

1.10　两个连续的 print()函数输出内容一般会分行显示，即调用 print()函数后会

换行并结束当前行，如何让两个 print()函数的输出打印在一行内？

1.11 import 保留字用来引入函数库，绘制图形可以使用什么 Python 函数库？

1.12 获得系统的日期和时间使用什么 Python 函数库？

1.5 程序的基本编写方法

要点： 每个程序都有统一的运算模式，即输入数据、处理数据和输出数据，这种朴素运算模式形成了程序的基本编写方法，即 IPO 方法。

1.5.1 IPO 程序编写方法

每个计算机程序都用来解决特定计算问题。较大规模的程序提供丰富的功能解决完整的计算问题，例如，控制航天飞机运行的程序、操作系统等；小型程序或程序片段可以为其他程序提供特定计算支持，作为解决更大计算问题的组成部分。无论程序规模如何，每个程序都有统一的运算模式：输入数据、处理数据和输出数据。这种朴素运算模式形成了基本的程序编写方法：IPO（Input, Process, Output）方法。

输入（Input）是一个程序的开始。程序要处理的数据有多种来源，因此形成了多种输入方式，包括文件输入、网络输入、控制台输入、交互界面输入、随机数据输入、内部参数输入等。

（1）文件输入：将文件作为程序输入来源。在获得文件控制权后，需要根据文件格式解析内部具体数据。例如，统计 Excel 文件数据的数量，需要首先获得 Excel 文件的控制权，打开文件后根据 Excel 中数据存储方式获得所需处理的数据，进而开展计算。7.1 节将具体介绍文件的使用。

（2）网络输入：将互联网上的数据作为输入来源。使用网络数据需要明确网络协议和特定的网络接口。例如，捕获并处理互联网上的数据，需要使用协议 HTTP 并解析 HTML 格式。第 10 章将介绍网络爬虫的原理和方法。

（3）控制台输入：将程序使用者输入的信息作为输入来源。当程序与用户间存在交互时，程序需要有明确的用户提示，辅助用户正确输入数据。从程序语法来说，这种提示不是必需的，但良好的提示设计有助于提高用户体验。

（4）交互界面输入：通过提供一个图形交互界面从用户处获得输入来源。此时，鼠标移动或单|双击操作、文本框内的键盘操作等都为程序提供输入的方式。

（5）随机数据输入：将随机数作为程序输入，这需要使用特定的随机数生成器程序或调用相关函数。4.5 节将详细介绍产生随机数的方法。

（6）内部参数输入：以程序内部定义的初始化变量为输入，尽管程序看似没有从外部获得输入，但程序执行之前的初始化过程为程序赋予了执行所需的数据。

输出（Output）是程序展示运算成果的方式。程序的输出方式包括控制台输出、

图形输出、文件输出、网络输出、操作系统内部变量输出等。

（1）控制台输出：以计算机屏幕为输出目标，通过程序运行环境中的命令行打印输出结果。这里“控制台”可以理解为启动程序的环境，例如，Windows 中的命令行工具、IDLE 工具等。

（2）图形输出：在计算机中启动独立的图形输出窗口，根据指令绘制运算结果。第 9 章将介绍高级人机交互方法。

（3）文件输出：以生成新的文件或修改已有文件方式输出运行结果，这是程序常用的输出方式。7.1 节将具体介绍文件的使用。

（4）网络输出：以访问网络接口方式输出数据。第 10 章将介绍自动向搜索引擎提交关键词查询的实例。

（5）操作系统内部变量输出：指程序将运行结果输出到系统内部变量中，这类变量包括管道、线程、信号量等。

处理（Process）是程序对输入数据进行计算产生输出结果的过程。计算问题的处理方法统称为“算法”，它是程序最重要的组成部分。可以说，算法是一个程序的灵魂。

——是否存在没有输入输出的程序呢？

——存在，例如，无限循环，代码如下：

```
1  while(True):
2      a = 1
```

这个无限循环程序包含两行语句，其中，while()根据括号内部值的真假决定是否进入循环，当括号内值为真时，进入第 2 行语句执行，否则跳过。由于括号内值被设定为 True（即真），代码将一直执行下去。

无限循环程序尽管没有输入也没有输出，它也有价值。通过不间断执行，该程序快速消耗 CPU 的计算资源，可以用来辅助测试 CPU 或系统性能。尽管如此，这类没有输入输出的程序在功能上十分有限，仅在特殊情况下使用。

IPO 不仅是程序设计的基本方法，也是描述计算问题的方式。以微实例 1.1 圆面积的计算为例，其 IPO 描述如下。

输入：圆半径 raduis

处理：计算圆面积 area = π * radius * radius

　　　此处，π 取 3.141 5

输出：圆面积 area

可以看到，问题的 IPO 描述实际上是对一个计算问题输入、输出和求解方式的自然语言描述，为了区别于其他描述方式，本书中所有 IPO 描述都包括“输入”、“处理”和“输出”3 个引导词。

IPO 描述能够帮助初学程序设计的读者理解程序设计的开始过程，即了解程序的运算模式，进而建立设计程序的基本概念。IPO 方法是非常基本的程序设计方法，随着学习深入，本书第 8 章将从设计思想入手介绍程序设计方法论，建立对较大规模程序框架的理解。

1.5.2　理解问题的计算部分

编写程序的目的是“使用计算机解决问题”。一般来说，“使用计算机解决问题”可以分为如下 6 个步骤。

（1）分析问题，分析问题的计算部分：首先必须明确，计算机只能解决计算问题，即解决一个问题的计算部分。清楚理解所需解决问题的计算部分十分重要，这是利用计算机解决问题的前提，对计算部分的不同理解会产生不一样的程序。

例如，本书每章节后都有若干道程序练习题，如果本书被当作教材，教师们可能会根据这些习题布置课后作业，此时，一些读者可能会思考这样的问题：如何由计算机辅助求解习题答案并完成作业？对这个问题的分析和理解可以有多个角度。

第一，对于作业中的数学计算，可以编写程序辅助完成，但利用哪些计算公式则由读者自己选择或设计。此时，该问题的计算部分表现为对某些数学公式的计算。

第二，可以利用互联网搜索课后练习题答案，根据搜索结果完成作业。为了降低网络答案错误的风险，可以通过计算机辅助获得多份答案并选择结果一致数量最多的答案作为“正确”答案。此时，该问题的计算部分表现为在网络上自动搜索多份结果并输出最可能的正确结果的过程。

第三，计算机是否可以直接理解课后练习题并给出答案呢？如果从这个角度出发，该问题的计算部分就表现为计算机对练习题的理解和人工智能求解。直到今天，具有高度智能的计算机仍然是全球科学家共同研究的目标。如果读者从这个角度分析该问题的计算部分，恐怕将无法在短时间内提交作业了。

拓展： 人工智能和图灵测试

人工智能（Artificial Intelligence，AI）是计算机科学的一个分支，区别于人类智能，人工智能指由机器或软件所体现的智能。“智能”这个概念难以确切定义，因此，1950 年，计算机科学之父艾伦·图灵（Alan Turing）提出了著名的“图灵测试”，从测试角度描述智能的含义。图灵测试中，机器和人分别通过文本途径（避免计算机理解语言能力不足的影响）回答一组由独立评判人提出的问题，如果评判人无法从回答中区分机器和人，则认为机器通过测试，具备与人相当的智能。图灵测试并不评判问题答案的正确性，而是通过评判答案之间的相似性确定机器是否具备智能。图灵测试是人工智能中的重要概念，也是奠定人工智能发展的重要评判基础。

这个例子说明，对一个问题中计算部分，不同理解将产生不同的计算问题，也将产生不同功能和复杂度的程序。如何更好地理解一个问题的计算部分，如何有效地利用计算机解决问题，这不只是编写程序的问题，而是更重要的思维问题，即计算思维。

（2）划分边界，划分问题的功能边界：计算机只能完成确定性的计算功能，因此，在分析问题计算部分的基础上，需要精确定义或描述问题的功能边界，即明确问题的输入、输出和对处理的要求。可以利用 IPO 方法辅助分析问题的计算部分，

给出问题的 IPO 描述。这个步骤只关心功能需求，无须关心功能的具体实现方法，需要明确程序的输入、输出以及输入输出之间的总体功能关系。

（3）设计算法，设计问题的求解算法：在明确处理功能的基础上，如何实现程序功能呢？这需要设计问题的求解算法。简单的程序功能中输入和输出间关系比较直观，程序结构比较简单，直接选择或设计算法即可。对于复杂的程序功能，需要利用程序设计方法将“大功能”划分成“小功能”，或者将功能中相对独立的部分封装成具备属性和操作的类，并在各功能或类之间设计处理流程。对于“小功能”或类中的操作，可以将它们看成一个新的计算问题，按照本节讲述的步骤逐级设计和实现。更多有关程序设计方法的内容将在第 8 章介绍。

（4）编写程序，编写问题的计算程序：选择一门编程语言，将程序结构和算法设计用编程语言来实现。原则上，任何通用编程语言都可以用来解决计算问题，在正确性上没有区别。然而，不同编程语言在程序的运行性能、可读性、可维护性、开发周期和调试等方面有很大不同。Python 语言相比 C 语言在运行性能上略有逊色，不适合性能要求十分苛刻的特殊计算任务；但 Python 程序在可读性、可维护性和开发周期等方面比 C 语言有更大优势。当代计算机经过了长期发展，性能远超过一般功能程序的使用需求，Python 语言在运行性能方面的微弱劣势对解决一般问题十分微不足道。

（5）调试测试，调试和测试程序：运行程序，通过单元测试和集成测试评估程序运行结果的正确性。一般来说，程序错误（通常称为 bug）与程序规模成正比。即使经验丰富的程序员编写的程序也会存在 bug，不同只在于 bug 数量的多少和发现的难易。为此，找到并排除程序错误十分必要，这个过程称为调试（通常称为 Debug）。

当程序正确运行后，可以采用更多测试发现程序在各种情况下的特点，例如，压力测试能够获得程序运行速度的最大值和稳定运行的性能边界，安全性测试能够发现程序漏洞，界定程序安全边界，进而指导程序在合理范围内使用。

（6）升级维护，适应问题的升级维护：任何一个程序都有它的历史使命，在这个使命结束之前，随着功能需求、计算需求和应用需求的不断变化，程序将不断地升级维护，以适应这些变化。

综上所述，解决计算问题包括 6 个步骤：分析问题、划分边界、设计算法、编写程序、调试测试和升级维护。其中，与程序设计语言和具体语法有关的步骤是编写程序和调试测试。可见，在解决计算问题过程中，编写程序只是一个环节。在此之前，分析问题、划分边界和设计算法都是重要步骤，经过这些步骤，一个计算问题已经能够在设计方案中被解决了，这个过程可以看作是计算思维的创造过程；编写程序和调试测试则是对解决方案的计算机实现，属于技术实现过程。

思考与练习

1.13　针对如下计算问题：测试一台机器是否真正拥有人类的智能。请用 IPO 方法描述该问题的解决方案。（参考图灵测试）

1.14　解决计算问题过程中，哪些步骤中可能用到 Python 语言？

1.15 调试和测试有什么区别和联系？

1.16 下面不是 IPO 模式的一部分的是（ B ）。

A．Input　　B．Program　　C．Process　　D. Output

1.6 Python 语言的版本更迭

要点：Python 2.x 已经是遗产，Python 3.x 是这个语言的现在和未来。

1.6.1 版本之间的区别

2010 年，Python 2.x 系列发布了最后一个版本，其主版本号为 2.7，同时，Python 维护者们声称不在 2.x 系列中继续进行主版本号升级。Python 2.x 系列已经完成了它的使命，逐步退出历史舞台。

2008 年，Python 3.x 第一个主版本发布，其主版本号为 3.0，并作为 Python 语言持续维护的主要系列。该系列在 2012 年推出 3.3 版本，2014 年推出 3.4 版本，2015 年推出 3.5 版本，2016 年推出 3.6 版本。目前，主要的 Python 标准库更新只针对 3.x 系列。

Python 3.x 是 Python 语言的一次重大升级，它不完全向下兼容 2.x 系列程序。在语法层面，3.x 系列继承了 2.x 系列绝大多数的语法表达，只是移除了部分混淆的表达方式。对于程序设计初学者来说，两者的差距很小，学会 3.x 系列也能看懂 2.x 系列语法。

学习 Python 语言不免会看到一些 2.x 系列的程序，为了让读者能看懂 2.x 系列程序代码，这里仅列出本书涉及内容中两个系列语法的一些区别（更多区别请查看 Guido 的文章：https://docs.python.org/3/whatsnew/3.0.html）。

（1）修改编码：3.x 系列默认采用 UTF-8 编码，因此处理中文与英文一样方便。而且，在表达 UTF-8 编码字符串时，不需要在前面增加 u 或者 U。

（2）修改 print 语句：用 print()函数替换了 print 语句，两者功能一样，格式不同，例如：

```
2.x: >>>print "The answer is", 2*2
3.x: >>>print("The answer is", 2*2)
2.x: >>>print x,
3.x: >>>print(x, end=",")
```

（3）修改 exec 语句：用 exec()函数替换了 exec 语句，两者功能一样，格式不同。

（4）去掉<>符号：用!=表示“不等于”。

（5）修改比较行为：用<、<=、>=、>符号比较两个元素时，如果元素之间不存

在有意义的顺序关系，将抛出 TypeError 错误，不再返回 False，例如：

```
2.x: >>>1 < "1"
False
3.x: >>>1 < "1"
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    1<"1"
TypeError: unorderable types: int() < str()
```

（6）去掉长整数类型：Python 3.x 系列不再区分整数和长整数类型，只有一个 int 类型，int 类型无取值范围限制。因此，sys.maxint 常量被去掉。

（7）修改整数除法：两个整数的一般除法（/）返回一个浮点数，不再返回一个整数，如果想返回整数，则用整数除法（//），例如：

```
2.x: >>>3/2
1
3.x: >>>3/2
1.5
3.x: >>>3//2
1
```

（8）修改八进制整数格式：使用 0o 开头，而不再使用 0 开头，例如，0o237，而不是 0237。

（9）增加关键字：增加 as、with、True、False、None 作为关键字。

（10）去掉 raw_input()函数：用 input()替代 raw_input()，input()返回一个字符串。

（11）修改 range()函数：range()功能与 Python 2.x 系列中 xrange()类似，不再显式返回一个列表。如果希望返回列表，需要通过 list()函数转换，例如：

```
2.x: >>>range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
3.x: >>>range(10)
range(0, 10)
3.x: >>>list(range(10))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

（12）修改返回类型：除 range()外，zip()、map()、filter()、字典类型的 key()方法、value()方法、item()方法不再返回列表类型。

（13）修改异常处理表达：使用 as 关键字标识异常信息，例如：

```
2.x: >>>try:
        ...    wrong_name
        ...except NameError, err:
        ...    print err
        ...
```

```
    name 'wrong_name' is not defined
3.x: >>>try:
    ...     wrong_name
    ...except NameError as err:
    ...     print(err)
    ...
name 'wrong_name' is not defined
```

拓展： 向后兼容

向后兼容（Backward Compatibility），也称为向下兼容，是系统、产品或技术的一个特性，用来描述一个新改进版本的系统、产品或技术继续同旧的、功能弱的版本一同工作的能力。简单说它指新版本可以兼容或者替代旧版本。对于Python 语言来说，Python 3.x 解释器不能不加修改地运行一个用 Python 2.x 语法编写的程序，Python 3.x 版本并不向后兼容 Python 2.x 版本。

向后兼容对产品和系统发展来说十分有益，因为用户可以无须了解产品升级的细节而使用新版本继续运行原有程序。因此，几乎所有重要系统或产品都向后兼容。例如，Intel 公司最新款 CPU 仍然能够运行最早期 80486 指令集编写的程序，微软公司最新的 Windows 操作系统能够运行早期 Windows XP 系统上的可执行文件。Python 语言不选择向后兼容，是 Guido 对 Python 语言发展做出的最重大的决定，虽然短期内带来升级函数库的巨大代价，但长期来看，由于不需要兼容旧有版本，新版本语言有助于简化解释器功能，释放 Python 语言发展的历史包袱。

1.6.2 版本的选择建议

——Python 学习者该学习哪个 Python 版本呢？

——除了一些特殊情况，请学习 Python 3.x 版本。

对于初次接触 Python 语言的读者，请学习 Python 3.x 系列版本。Python 版本更迭已达 8 年以上，目前，全部的标准库和绝大多数第三方库都很好地支持 Python 3.x 系列，并在该系列基础上升级更新。

尽管现在大多数 Linux 和 Mac OS 系统在发布中仍然默认集成 Python 2.x 系列版本，但 Python 3.x 系列已经非常成熟和稳定。部分 Linux 和 Mac OS 系统通过 Python3 命令同时提供对 Python 3.x 系列的支持。

但是，在遇到以下问题时，请考虑使用 Python 2.x 版本。

（1）所面对的开发环境已经部署好，并且采用 Python 2.x，在无法选择开发环境的情况下，请使用 Python 2.x 版本。

（2）如果希望使用一个特定的第三方库，且这个库不提供 Python 3.x 版本（一般来说，这类库都已经多年无人维护），请使用 Python 2.x 版本。

总的来说，如果可以自主选择版本且所使用的库或已有代码有 Python 3.x 的支持，请选择 Python 3.x 版本。基于上述考虑，本书以 Python 3.x 为教学内容，也请读者选择 Python 3.x 版本。

思考与练习

1.17　打印输出（print）是程序常用功能，观察 Python 2.x 和 Python 3.x 版本在这个功能上的不同。

1.18　获得用户输入（input）也是程序常用功能，观察 Python 2.x 和 Python 3.x 版本在这个功能上的不同。

1.19　如何快速判断一个 Python 代码是 Python 3.x 版本。

阶段测试 1-1：Python 偶遇小测验



本 章 小 结

本章具体讲解了计算机的基本定义、计算机的功能性和可编程性、程序设计语言分类、编译和解释、Python 语言的历史和发展、配置 Python 开发环境等内容，最后给出了 Python 版本的主要区别供读者参考。

程序练习 1-1：十分钟学 Python



程序练习题

由于本章尚未开始讲解 Python 语言语法，请读者在 Python 3.x 环境中运行下列程序了解 Python 语言，熟练掌握 Python 语言的开发和运行环境。

源代码 1-7：字符串拼接



1.1　字符串拼接。接收用户输入的两个字符串，将它们组合后输出。

```
1  str1 = input("请输入一个人的名字：")
2  str2 = input("请输入一个国家名字：")
3  print("世界这么大，{}想去{}看看。".format(str1,str2))
```

源代码 1-8：整数序列求和



1.2　整数序列求和。用户输入一个正整数 N，计算从 1 到 N（包含 1 和 N）相加之后的结果。

```
1  n = input("请输入整数 N: ")
2  sum = 0
3  for i in range(int(n)):
4      sum += i + 1
5  print("1 到 N 求和结果: ", sum)
```

源代码 1-9：九九乘法表输出



1.3 九九乘法表输出。工整打印输出常用的九九乘法表，格式不限。

```
for i in range(1,10):
    for j in range(1,i+1):
        print("{}*{}={:2} ".format(j,i,i*j), end='')
    print('')
```

源代码 1-10：阶乘求和



1.4 计算 1+2!+3!+⋯+10!的结果。

```
sum, tmp = 0, 1
for i in range(1,11):
    tmp*=i
    sum+=tmp
print("运算结果是：{}".format(sum))
```

源代码 1-11：猴子吃桃问题



1.5 猴子吃桃问题：猴子第一天摘下若干个桃子，当即吃了一半，还不过瘾，又多吃了一个；第二天早上又将剩下的桃子吃掉一半，又多吃了一个。以后每天早上都吃了前一天剩下的一半多一个。到第五天早上想再吃时，见只剩下一个桃子了。请编写程序计算猴子第一天共摘了多少桃子。

```
n = 1
for i in range(5,0,-1):
    n = (n+1)<<1
print(n)
```

源代码 1-12：健康食谱输出



1.6 健康食谱输出。列出 5 种不同食材，输出它们可能组成的所有菜式名称。

```
diet = ['西红柿', '花椰菜', '黄瓜', '牛排', '虾仁']
for x in range(0, 5):
    for y in range(0, 5):
        if not(x == y):
            print("{}{}".format(diet[x], diet[y]))
```

源代码 1-13：五角星绘制



1.7 五角星的绘制：绘制一个红色的五角星图形，如图 1.11 所示。

```
from turtle import *
fillcolor("red")
begin_fill()
while True:
    forward(200)
    right(144)
    if abs(pos()) < 1:
        break
end_fill()
```

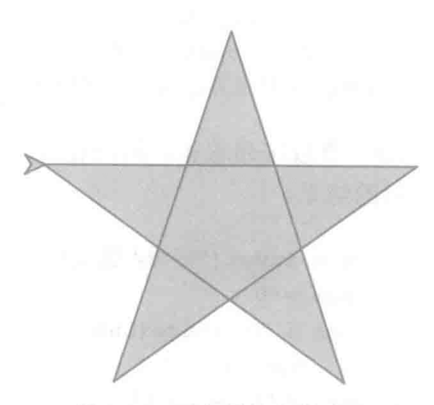


图 1.11 五角星的绘制结果

1.8　太阳花的绘制：绘制一个太阳花的图形，如图 1.12 所示。




```
from turtle import *
color('red', 'yellow')
begin_fill()
while True:
    forward(200)
    left(170)
    if abs(pos()) < 1:
        break
end_fill()
done()
```

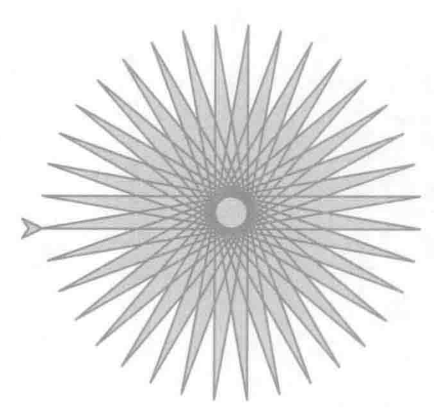


图 1.12　太阳花的绘制结果

第 2 章　Python 程序实例解析

电子教案 2-1：Python 程序实例解析



代码胜于雄辩。

Talk is cheap. Show me the code.

——林纳斯·托瓦兹 （Linus Torvalds）

Linux 操作系统的奠基者

学习目标

（1）掌握解决计算问题的一般方法。

（2）掌握 Python 语言的基本语法，包括缩进、变量、命名等。

（3）掌握 Python 语言绘制图形的一般方法。

（4）了解 Python 标准库的导入和使用。

Python 创始人 Guido 大牛是英国 Monty Python 六人组合的忠实粉丝，因此，他将创建的编程语言起名为 Python。尽管 Guido 起名的本意与蟒蛇无关，但世界各地忠实的 Python 程序员仍将蟒蛇当作该语言的吉祥物。本章将为读者讲解如何使用 Python 标准库快速绘制一条能够爬行的彩色大蟒蛇。

向 Guido 致敬、向创新致敬！

2.1 实例 1：温度转换

要点： 这是一个程序设计教学中的经典实例，用于理解基本的 Python 语法元素。

本节以温度转换问题为例，介绍程序设计的基本方法，并给出 Python 语言的具体实现。

温度的刻画有两个不同体系：摄氏度（Celsius）和华氏度（Fabrenheit）。摄氏度以 1 标准大气压下水的结冰点为 0 度，沸点为 100 度，将两个温度区间进行 100 等分后确定 1 度所代表的温度区间，进而刻画温度值。华氏度以 1 标准大气压下水的结冰点为 32 度，沸点为 212 度，将两个温度区间进行 180 等分后定义为 1 度区间。华氏度的 1 度比摄氏度的 1 度所对应温度区间更小，所以华氏度体系更为细致。由于历史原因，不同国家可能采用不同的温度表示方法，例如，中国采用摄氏度，美国采用华氏度。

对于去美国旅行的中国游客来说，需要将当地发布的华氏温度转换为摄氏温度以符合自己的理解习惯；同样，来中国旅行的美国游客，也需要将当地发布的摄氏温度转换为华氏温度。问题是，如何利用计算机程序辅助旅行者进行温度转换？

根据第 1 章介绍的程序编写基本方法，用计算机解决上述问题需要 6 个步骤，分析和实现过程如下。

（1）分析问题：可以从很多不同角度来理解旅行者温度转换问题的计算部分。这里给出 3 个角度。第一，利用程序进行温度转换，由用户输入温度值，程序给出输出结果。这是最直观的理解。第二，可以通过语音识别、图像识别等方法自动监听并获得温度信息发布渠道（如收音机、电视机等）给出的温度播报源数据，再由程序转换后输出给用户。这种角度相比第一种不需要用户给出输入。第三，随着互联网的高度普及和接入的便捷，程序也可以定期从温度信息发布网站获得温度值，再将温度信息转换成旅行者熟悉的方式。3 种角度对问题计算部分的不同理解会产生不同的 IPO 描述、算法和程序。应该说，“利用计算机解决问题”需要结合计算机技术的发展水平和人类对问题的思考程度，在特定技术和社会条件下，分析出一个问题最经济、最合理的计算部分，进而用程序实现。本文以第一种理解角度为例编写并讲解余下程序步骤。

（2）划分边界：在确定问题计算部分的基础上进一步划分问题边界，即明确问题的输入数据、输出数据和对数据处理的要求。由于程序可能接收华氏温度和摄氏温度，并相互转换，该功能的 IPO 描述如下。

输入：带华氏或摄氏标志的温度值

处理：根据温度标志选择适当的温度转换算法

输出：带摄氏或华氏标志的温度值

这里采用 82F 表示华氏 82 度，采用 28C 表示摄氏 28 度，实数部分是温度值。这种温度表示格式同时用于温度的输入和输出。

（3）设计算法：根据华氏和摄氏温度定义，两个温度体系都以 1 标准大气压下水的结冰点和沸点为温度区间边界，因此，转换算法如下：

```
C = (F - 32)/1.8
F = C*1.8 + 32
```

其中，C 表示摄氏温度，F 表示华氏温度。

拓展： 算法

算法（Algorithm）是数学和计算领域的概念，指完成特定计算的一组有序操作。在 IPO 模式中，确定输入和输出后，处理过程通常也称为算法。算法的理解有广义和狭义之分。广义上，任何完成计算功能的一组操作都可以称为算法，这组操作可以对应单一的计算问题，也可以对应多个计算问题的组合；狭义上，算法通常针对单一计算问题，例如，针对优化求解问题的贪婪算法、遍历树结构的深度优先算法等。

（4）编写程序：根据 IPO 描述和算法设计，编写如下温度转换的 Python 程序代码：

源代码 2-1：基本的温度转换程序



实例代码 1.1 e1.1TempConvert.py

```
1   #e1.1TempConvert.py
2   TempStr = input("请输入带有符号的温度值：")
3   if TempStr[-1] in ['F','f']:
4       C = (eval(TempStr[0:-1]) - 32)/1.8
5       print("转换后的温度是{:.2f}C".format(C))
6   elif TempStr[-1] in ['C','c']:
7       F = 1.8*eval(TempStr[0:-1]) + 32
8       print("转换后的温度是{:.2f}F".format(F))
9   else:
10      print("输入格式错误")
```

此时看不懂上述代码没关系，2.2 节将逐行解释上述代码的含义。

（5）调试测试：将上述程序保存为文件：e1.1TempConvert.py，采用 1.4 节介绍的方法，使用 IDLE 运行该程序。输出和交互输入如下[1]。

输入带华氏标志的温度值，程序运行结果如下：

```
>>>
请输入带有符号的温度值：82F
转换后的温度是27.78C
```

[1] 本书中，交互代码中的黑体字表示用户输入信息。

输入带摄氏标志的温度值，程序运行结果如下：

```
>>>
请输入带有符号的温度值：-30C
转换后的温度是-22.00F
```

上述程序符合 Python 语法，执行结果正确。事实上，当程序较为复杂时，很难保证一次编写后的程序能够直接正确运行或运行逻辑没有错误。甚至说，任何程序都会有错误。寻找错误的调试过程不容忽视。

（6）升级维护：与人一样，任何程序都有生命周期。促使程序生命结束的事件有很多，例如，平台更换、使用方式变化、算法改进等。对于上述例子，只要中国、美国使用不同的温度标准，温度转换问题将一直存在。随着问题使用场景、输入和输出要求等因素的变化，程序将需要不断地维护和升级。

思考与练习

2.1　公司或组织都需要对资金使用进行管理，因此需要计算机辅助进行财务统计和报表分析。请从不少于 3 个角度分析该问题的计算部分。

2.2　《红楼梦三国演义》是中国四大名著之一，该书描述了 100 多个典型人物。统计书中典型人物名字出现的次数能够侧面反映人物的重要性。请给出这个计算问题的 IPO 描述，重点描述其中的算法部分。6.6 节将给出统计《三国演义》中人物出场次数的程序。

2.3　程序设计不能解决所有问题。例如，计算机无法回答如下这些问题：你最欣赏的历史人物是谁？孙红雷和姚晨两位演员，谁的演技更好？创新对中国未来经济的价值有多大？请讨论总结，哪些类型的问题无法通过程序设计解决？

2.2　Python 程序语法元素分析

要点： Python 程序包括格式框架、注释、变量、表达式、分支语句、循环语句、函数等语法元素。

图片资料 2-1：Python 快速参考之 Python 基础语法



程序设计的 6 个步骤是利用计算机解决问题的方法步骤，程序设计语言则是解决问题的实现载体。本节以实例代码 1.1 为例，介绍 Python 程序中各组成部分，即语法元素的基本含义，使读者对 Python 程序有一个基本的理解。各元素的深入介绍将在第 3 章到第 7 章展开。

2.2.1　程序的格式框架

Python 语言采用严格的“缩进”来表明程序的格式框架。缩进指每一行代码开

始前的空白区域，用来表示代码之间的包含和层次关系。不需要缩进的代码顶行编写，不留空白。代码编写中，缩进可以用 Tab 键实现，也可以用多个空格（一般是 4 个空格）实现，但两者不混用。建议采用 4 个空格方式书写代码。

严格的缩进可以约束程序结构，有利于维护代码结构的可读性。例如，在实例代码 1.1 的 10 行代码中，第 4、5、7、8、10 行存在缩进，表明这些行代码在逻辑上属于之前紧邻的无缩进代码行的所属范畴。

图 2.1(a)给出了实例代码 1.1 的缩进关系，其中箭头表示 if 语句与后面语句之间单层缩进关系。除了单层缩进，一个程序的缩进还可“嵌套”从而形成多层缩进，图 2.1(b)给出了 4.6 节中实例代码 6.1 的多层缩进关系。Python 语言对语句之间的层次关系没有限制，可以“无限制”嵌套使用。

```
#e1.1TempConvert.py
TempStr = input("请输入带有符号
if TempStr[-1] in ['F','f']:
    C = (eval(TempStr[0:-1]) -
    print("转换后的温度是{:.2f}C
elif TempStr[-1] in ['C','c']
    F = 1.8*eval(TempStr[0:-1]
    print("转换后的温度是{:.2f}F
else:
    print("输入格式错误")
```

(a) 单层缩进

```
DARTS = 1000
hits = 0.0
clock()
for i in range(1, DARTS):
    x, y = random(), random(
    dist = sqrt(x ** 2 + y *
    if dist <= 1.0:
        hits = hits + 1
pi = 4 * (hits/DARTS)
print("Pi的值是{:.2f}".format
```

(b) 多层缩进

图 2.1　Python 程序的格式框架

缩进表达了所属关系。单层缩进代码属于之前最邻近的一行非缩进代码，多层缩进代码根据缩进关系决定所属范围。需要注意，不是所有代码都可以通过缩进包含其他代码，图 2.1(a)所示的缩进代码包含在 if-elif-else 这种判断结构中。一般来说，判断、循环、函数、类等语法形式能够通过缩进包含一批代码，进而表达对应的语义。但是，如 print()这样的简单语句不表达包含关系，不能使用缩进。

2.2.2　注释

注释是程序员在代码中加入的一行或多行信息，用来对语句、函数、数据结构或方法等进行说明，提升代码的可读性。注释是辅助性文字，会被编译或解释器略去，不被计算机执行。例如，实例代码 1.1 中第 1 行就是一个注释。

```
1    #e1.1TempConvert.py
```

Python 语言有两种注释方法：单行注释和多行注释。单行注释以#开头，多行注释以'''（3 个单引号）开头和结尾。例如：

```
1  # 这是单行注释，单行注释可以独占一行
2  print(pow(2,10)) # 计算 2 的 10 次方，单行注释可以从行的中间开始
3  '''
4  print(pow(2,10)) 此行是注释，不被计算机执行
5  此行也是注释
6  '''
```

Python 程序中的非注释语句将按顺序执行，而注释语句则被解释器过滤掉，不被执行。本书中完整实例程序的首行都会有一个注释行，用来说明该程序保存为文件时建议采用的名字，读者可以从本书电子资源中获取同名代码文件。

注释主要有 3 个用途。第一，标明作者和版权信息。在每个源代码文件开始前增加注释，标记编写代码的作者、日期、用途、版权声明等信息，可以采用单行或多行注释。第二，解释代码原理或用途。在程序关键代码附近增加注释，解释关键代码作用，增加程序的可读性。由于程序本身已经表达了功能意图，为了不影响程序阅读连贯性，程序中的注释一般采用单行注释，标记在关键代码同行。对于一段关键代码，可以在其附近采用一个多行注释或多个单行注释给出代码设计原理等信息。第三，辅助程序调试。在调试程序时，可以通过单行或多行注释临时“去掉”一行或连续多行与当前调试无关的代码，辅助程序员找到程序发生问题的可能位置。

2.2.3 命名与保留字

与数学概念类似，Python 程序采用“变量”来保存和表示具体的数据值。为了更好地使用变量等其他程序元素，需要给它们关联一个标识符（名字），关联标识符的过程称为命名。命名用于保证程序元素的唯一性。例如，实例代码 1.1 中，TempStr 是一个接收输入字符串的变量名字。

Python 语言允许采用大写字母、小写字母、数字、下画线_和汉字等字符及其组合给变量命名，但名字的首字符不能是数字，中间不能出现空格，长度没有限制[1]。以下是合法命名的标识符：

```
python_is_good、python_is_not_good、_is_it_a_question_
```

喜欢 Python 语言、我喜欢这本 Python 书籍

注意：标识符对大小写敏感，python 和 Python 是两个不同的名字。

一般来说，程序员可以为程序元素选择任何喜欢的名字，但这些名字不能与 Python 的保留字相同。Python 3.x 版本共有 33 个保留字，如表 2.1 所示。与其他标识符一样，Python 的保留字也对大小写敏感。例如，for 是保留字，而 For 则不是，程序员可以定义其为变量使用。

[1] 实际上，受限于计算机存储资源，Python 语言中定义的名字是有长度限制的，但这种限制只是计算机资源层面的限制，从语法上没有限制。

拓展： 保留字

保留字（Keyword），也称为关键字，指被编程语言内部定义并保留使用的标识符。程序员编写程序时不能定义与保留字相同的标识符。每种程序设计语言都有一套保留字，保留字一般用来构成程序整体框架、表达关键值和具有结构性的复杂语义等。掌握一门编程语言首先要熟记其所对应的保留字。

Python 3 系列可以采用中文等非英语语言字符对变量命名。由于存在输入法切换、平台编码支持、跨平台兼容等问题，从编程习惯和兼容性角度考虑，一般不建议采用中文等非英语语言字符对变量命名。本书所有程序的变量命名都采用英文字符，由于注释内容不被解释器执行，本书注释内容采用中文描述。

表 2.1　Python 3 的 33 个保留字列表

False	def	if	raise
None	del	import	return
True	elif	in	try
and	else	is	while
as	except	lambda	with
assert	finally	nonlocal	yield
break	for	not	
class	from	or	
continue	global	pass	

2.2.4　字符串

存储和处理文本信息在计算机应用中十分常见。文本在程序中用字符串（string）类型来表示。Python 语言中，字符串是用两个双引号" "或者单引号' '括起来的一个或多个字符。实例代码 1.1 中第 2、3、5、6、8、10 行代码都包含字符串。

字符串是字符的序列，可以按照单个字符或字符片段进行索引。字符串包括两种序号体系：正向递增序号和反向递减序号，如图 2.2 所示。如果字符串长度为 L，正向递增以最左侧字符序号为 0，向右依次递增，最右侧字符序号为 L-1；反向递减序号以最右侧字符序号为–1，向左依次递减，最左侧字符序号为–L。这两种索引字符的方法可以同时使用。实例代码 1.1 中第 3 行 TempStr[-1]表示字符串 TempStr 变量的最后一个字符。



图 2.2　Python 字符串的两种序号体系

Python 字符串也提供区间访问方式，采用[N: M]格式，表示字符串中从 N 到 M（不包含 M）的子字符串，其中，N 和 M 为字符串的索引序号，可以混合使用正向

递增序号和反向递减序号。实例代码 1.1 中第 4、7 行的 TempStr[0:-1]表示字符串 TempStr 变量第 0 个字符开始到最后一个字符（但不包含最后一个字符）的子串。

Python 语言有丰富的字符串处理方法，3.5 节将详细介绍。这里，以温度转换实例中的语句为例，假如用户输入的字符串是"110C"，相应的字符串操作结果如下：

```
>>>TempStr = "110C"
>>>print(TempStr[-1])
C
>>> print(TempStr[0:-1])
110
```

2.2.5 赋值语句

程序中产生或计算新数据值的代码称为表达式，类似数学中的计算公式。表达式以表达单一功能为目的，运算后产生运算结果，运算结果的类型由操作符或运算符决定，如实例代码 1.1 中第 2、4、7 等行都包含表达式。

Python 语言中，“=”表示“赋值”，即将等号右侧的计算结果赋给左侧变量，包含等号（=）的语句称为赋值语句。实例代码 1.1 第 2 行表示将等号右侧 input()函数的结果赋值给左侧变量 TempStr。

此外，还有一种同步赋值语句，可以同时给多个变量赋值，基本格式如下：

```
<变量 1>,…, <变量 N> = <表达式 1>,…, <表达式 N>
```

同步赋值并非等同于简单地将多个单一赋值语句进行组合，因为，Python 在处理同步赋值时首先运算右侧的 N 个表达式，同时将表达式的结果赋值给左侧 N 个变量。例如，互换变量 x 和 y 的值，如果采用单一语句，需要一个额外变量辅助，代码如下：

```
>>>t = x
>>>x = y
>>>y = t
```

如果采用同步赋值，一行语句即可：

```
>>>x, y = y, x
```

同步赋值语句可以使赋值过程变得更简洁，通过减少变量使用，简化语句表达，增加程序的可读性。但是，应尽量避免将多个无关的单一赋值语句组合成同步赋值语句，否则会降低程序可读性。那么，如何判断多个单一赋值语句是否相关呢？一般来说，如果多个单一赋值语句在功能上表达了相同或相关的含义，或者在程序中属于相同的功能，都可以采用同步赋值语句。

2.2.6 input()函数

实例代码 1.1 中的第 2 行使用了一个 input()函数从控制台获得用户输入，无论

用户在控制台输入什么内容，input()函数都以字符串类型返回结果。

```
2  TempStr = input("请输入带有符号的温度值: ")
```

在获得用户输入之前，input()函数可以包含一些提示性文字，使用方法如下：

```
<变量> = input(<提示性文字>)
```

需要注意，无论用户输入的是字符或是数字，input()函数统一按照字符串类型输出。在如下例子中，当用户输入数字 1 024.256 时，input()函数以字符串形式输出。

```
>>>input("请输入: ")
请输入: python
'python'
>>> input("请输入: ")
请输入: 1024.256
'1024.256'
```

2.2.7　分支语句

分支语句是控制程序运行的一类重要语句，它的作用是根据判断条件选择程序执行路径，使用方式如下：

```
if <条件 1>:
    <语句块 1>
elif <条件 2>:
    <语句块 2>
…
else:
    <语句块 N>
```

其中，if、elif、else 都是保留字，else 后面不增加条件，表示不满足其他 if 语句的所有其余情况。实例代码 1.1 中第 3、6、9 行采用了“if-elif-else”类型的分支语句，如下：

```
3  if TempStr[-1] in ['F','f']:
6  elif TempStr[-1] in ['C','c']:
9  else:
```

其中，第 3 行 if 语句包含第一个条件表达式：

```
TempStr[-1] in ['F', 'f']
```

该表达式由保留字 in 组成，表示判断字符串 TempStr 的最后一个字符（TempStr[-1]）是否在一个由'F'或者'f'组成的集合中，即 TempStr[-1]是否等于'F'或者'f'。如果相等，则返回 True，否则返回 False。

对于 if 语句来说，当 in 表达式返回 True 时，执行第 4、5 行语句，如果返回 False，则执行第 6 行的 elif 语句，判断下一个条件。第 3 行语句在程序语义上是判

断用户输入的温度值是否是华氏度。

同理，第 6 行 elif 语句判断字符串 TempStr 的最后一个字符（TempStr[-1]）是否在一个由'C'或者'c'组成的集合中，如果条件成立，则继续执行第 7、8 行语句，否则执行第 9 行语句。第 6 行语句在程序语义上是判断用户输入的温度值是否是摄氏度。

第 9 行 else 语句没有判断条件，表示当所有 if、elif 条件都不满足时所执行的语句。该语句表示用户输入的内容不符合预定义的摄氏温度值或华氏温度值格式，对于该程序来说，用户输入错误。

第 3、6 行语句中用方括号和逗号组成的类型叫列表，其格式为：[元素 1, 元素 2,…,元素 *n*]。列表类型在程序设计中十分常用，6.2 节将详细介绍列表类型的使用。4.2 节将详细介绍有关分支的更多使用方法。

2.2.8 eval()函数

实例代码 1.1 中第 4、7 行是赋值语句，如下所示，其实现了 IPO 描述中两个温度体系的具体转换公式。这两行语句中包含了 eval()函数。

```
4   C = (eval(TempStr[0:-1]) - 32)/1.8
7   F = 1.8*eval(TempStr[0:-1]) + 32
```

eval(<字符串>)函数是 Python 语言中一个十分重要的函数，它能够以 Python 表达式的方式解析并执行字符串，并将返回结果输出。例如：

```
>>>x = 1
>>>eval("x + 1")
2
>>>eval("1.1 + 2.2")
3.3
```

简单说，eval(<字符串>)的作用是将输入的字符串转变成 Python 语句，并执行该语句。实例代码 1.1 使用 eval()函数将用户的部分输入（TempStr[0:-1]）由字符串转换成数字，假设用户输入**"102C"**，经过 eval()函数处理，将变成 Python 内部可进行数学运算的数值 102。

```
>>>TempStr = "102C"
>>>eval(TempStr[0:-1])
102
```

使用 eval()函数处理字符串需要注意合理使用。例如，如果直接输入字符串"hello"，eval()函数将去掉两个引号，将其解释为一个变量，由于之前没有定义过 hello 变量，解释器会报错。当输入字符串"'hello'"时，eval()函数去掉外部双引号后，内部还有一个引号，则'hello'被解释为字符串。eval()函数还有很多作用，请读者在实践中逐步挖掘。

```
>>>eval("hello")
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    eval("hello")
  File "<string>", line 1, in <module>
NameError: name 'hello' is not defined
>>>eval("'hello'")
'hello'
```

如果用户希望输入一个数字（小数或负数），并用程序对这个数字进行计算，可以采用 eval(input(<输入提示字符串>))的组合，例如：

```
>>>value = eval(input("请输入要计算的数值: "))
请输入要计算的数值: 1024.256
>>>print(value*2)
2048.512
```

实例代码 1.1 中第 4、7 行表达式中，等号右侧进行了算术运算。Python 支持+、-、*、/ 和**（幂）5 种基本算术运算操作。表达式右侧的含义是将 TempStr 字符串中除最后一位外的子串转换成数字，再对数字进行减法和除法运算。

Python 语法允许在表达式内部标记之间增加空格，这些多余的空格将被解释器去掉。下面这个语句与第 4 行语句功能一致。

```
4 | C = ( eval ( TempStr [ 0 : -1 ] ) - 32) / 1.8
```

适度增加空格有助于提高代码可读性，但要注意不能改变与缩进相关的空格数量，也不能在变量名等命名中间增加空格。

Python 语言的括号与数学运算中的括号一样，用来表示分组和优先级。不使用括号时，优先级按照算术优先级来确定，使用的多余括号将被编译程序去掉，不影响程序正确运行。下面语句与第 4 行语句功能一致。

```
4 | C = ( eval ( TempStr [(0):(-1)] ) - 32) / (1.8)
```

2.2.9　print()函数

实例代码 1.1 中第 5、8、10 行使用 print(<待输出字符串>)输出函数输出字符信息，其也能以字符形式输出变量。当输出纯字符信息时，可以直接将待输出内容传递给 print()函数，如第 10 行。当输出变量值时，需要采用格式化输出方式，通过 format()方法将待输出变量整理成期望输出的格式，如第 5、8 行。

```
5  | print("转换后的温度是{:.2f}C".format(C))
8  | print("转换后的温度是{:.2f}F".format(F))
10 | print("输入格式错误")
```

具体来说，print()函数用槽格式和 format()方法将变量和字符串结合到一起输出。例如第 5 行，输出的模板字符串是**"转换后的温度是{:.2f}C"**，其中大括号{}表示一个槽位置，这个括号中的内容由字符串后面紧跟的 format()方法中的参数 C 填充。大括号**{:.2f}**中的内容表示变量 C 输出的格式，具体表示输出数值取两位小数值，读者可以暂时不用深究，3.6 节将详细介绍字符串格式化输出方法。用两个小例子感受一下这段程序的魅力吧。

```
>>>C1, C2 = 10, 10.24024
>>>print("转换后的温度是{:.2f}C".format(C1))
转换后的温度是 10.00C
>>>print("转换后的温度是{:.2f}C".format(C2))
转换后的温度是 10.24C
```

2.2.10 循环语句

循环语句是控制程序运行的一类重要语句，与分支语句控制程序执行类似，它的作用是根据判断条件确定一段程序是否再次执行一次或者多次。

实例代码 1.1 不包含循环语句，程序执行一次后退出。如果希望程序一直运行，连续接受用户输入，直到用户输入的最后一个字符是'N'或'n'时退出，可以采用循环语句改造程序，如程序代码 1.2 所示。

源代码 2-2：循环执行的温度转换程序

实例代码 1.2 e1.2TempConvert.py

```
1   #e1.2TempConvert.py
2   TempStr = input("请输入带有符号的温度值: ")
3   while TempStr[-1] not in ['N','n']:
4       if TempStr[-1] in ['F','f']:
5           C = (eval(TempStr[0:-1]) - 32)/1.8
6           print("转换后的温度是{:.2f}C".format(C))
7       elif TempStr[-1] in ['C','c']:
8           F = 1.8*eval(TempStr[0:-1]) + 32
9           print("转换后的温度是{:.2f}F".format(F))
10      else:
11          print("输入格式错误")
12      TempStr = input("请输入带有符号的温度值: ")
```

循环语句有多种类型，实例代码 1.2 采用了条件循环。条件循环的基本过程如下：

```
while (<条件>):
    <语句块 1>
<语句块 2>
```

当条件为真（True）时，执行语句块 1 语句，这些语句通过缩进表达与 while

语句的所属关系。当条件为假（False）时，退出循环，执行循环后语句块 2 语句。

实例代码 1.2 第 3 行使用了条件循环，该循环条件用于判断用户输入的最后一个字符（TempStr[-1]）是否为**'N'**或**'n'**。

```
3  while TempStr[-1] not in ['N','n']:
```

如果该字符是**'N'**或者**'n'**，则条件语句结果为 False，退出循环，进而结束程序；否则，条件语句结果为 True，继续执行循环内部语句。这行语句中的 not 是保留字，表示对判断结果取反。4.4 节将详细介绍循环语句及其使用方法。

2.2.11 函数

实例代码 1.1 和实例代码 1.2 都是由一个序列表达式组成，程序按照顺序方式从头到尾执行。实际编程中，一般将特定功能代码编写在一个函数里，便于阅读和复用，也使得程序模块化更好。函数可以理解为对一组表达特定功能表达式的封装，它与数学函数类似，能够接收变量并输出结果。input()、print()、eval()都是 Python 解释器的内置函数。经过函数改造后的温度转换程序如实例代码 1.3 所示。

源代码 2-3：
函数封装的温度
转换程序



实例代码 1.3　　　　e1.3TempConvert.py

```
1   #e1.3TempConvert.py
2   def tempConvert(ValueStr):
3       if ValueStr[-1] in ['F','f']:
4           C = (eval(ValueStr[0:-1]) - 32)/1.8
5           print("转换后的温度是{:.2f}C".format(C))
6       elif ValueStr[-1] in ['C','c']:
7           F = 1.8*eval(ValueStr[0:-1]) + 32
8           print("转换后的温度是{:.2f}F".format(F))
9       else:
10          print("输入格式错误")
11  TempStr = input("请输入带有符号的温度值: ")
12  tempConvert(TempStr)
```

实例代码 1.3 第 2 行用 def 保留字定义了一个名为 tempConvert()的函数，它使用一个参数 ValueStr。tempConvert()函数所属代码是第 2 行后与之有缩进关系的代码，即第 3 到第 10 行。在这些代码中，ValueStr 变量作为输入函数的字符串使用。由 def 保留字定义的函数在程序中不被直接执行，需要使用函数名称调用才能执行。

由于第 11 行没有缩进，它与第 2 行是平行关系，程序第 1 行到第 10 行不直接执行，而从第 10 行开始执行，并接收用户输入存到变量 TempStr 中。第 12 行调用 tempConvert()函数，并将 TempStr 当作参数传递给函数的内部变量 ValueStr。接下来，程序根据 tempConvert()函数定义执行函数内容，完成温度转换功能。

简单地说，程序代码 1.3 通过 def 语句定义了 tempConvert()函数，并将原有功

程序练习 2-1：半小时学 Python



能封装在这个函数中，语句调用 tempConvert()函数执行这些功能。函数是代码编写中最重要的封装方式，可以辅助代码按照功能划分模块，有利于代码之间进行语句块级别的复用。第 5 章将介绍与函数有关的更多内容。

思考与练习

2.4 下面不符合 Python 语言命名规则的是（ ）。

A. monthly B. monTHly C. 3monthly D. _Monthly3_

阶段测试 2-1：Python 语法元素小测验



2.5 请写出 Python 语言的 33 个保留字，标记本书已经介绍过的保留字，并解释这些保留字的基本含义。

2.6 请用一行代码编写一个回声程序，将用户输入的内容直接打印出来。

2.7 试想一下，为什么 Python 的命名不能以数字开头？

2.3 实例 2：Python 蟒蛇绘制

> 要点：这是一个有趣的 Python 实例，用于理解 Python 的“模块编程”思想。

Python 英文是“蟒蛇”的意思，因此，绘制一条蟒蛇十分有趣。本节以“Python 蟒蛇绘制”为例，介绍使用 Python 绘制图形程序的基本方法，并讲解 Python 语言的“模块编程”思想。

实例代码 2.1 是“Python 蟒蛇绘制”的源代码，图 2.3 是该程序的输出效果。

源代码 2-4：Python 蟒蛇绘制程序



实例代码 2.1 e2.1DrawPython.py

```
1  #e2.1DrawPython.py
2  import turtle
3  turtle.setup(650, 350, 200, 200)
4  turtle.penup()
5  turtle.fd(-250)
6  turtle.pendown()
7  turtle.pensize(25)
8  turtle.pencolor("purple")
9  turtle.seth(-40)
10 for i in range(4):
11     turtle.circle(40, 80)
12     turtle.circle(-40, 80)
13 turtle.circle(40, 80/2)
14 turtle.fd(40)
15 turtle.circle(16, 180)
16 turtle.fd(40 * 2/3)
```

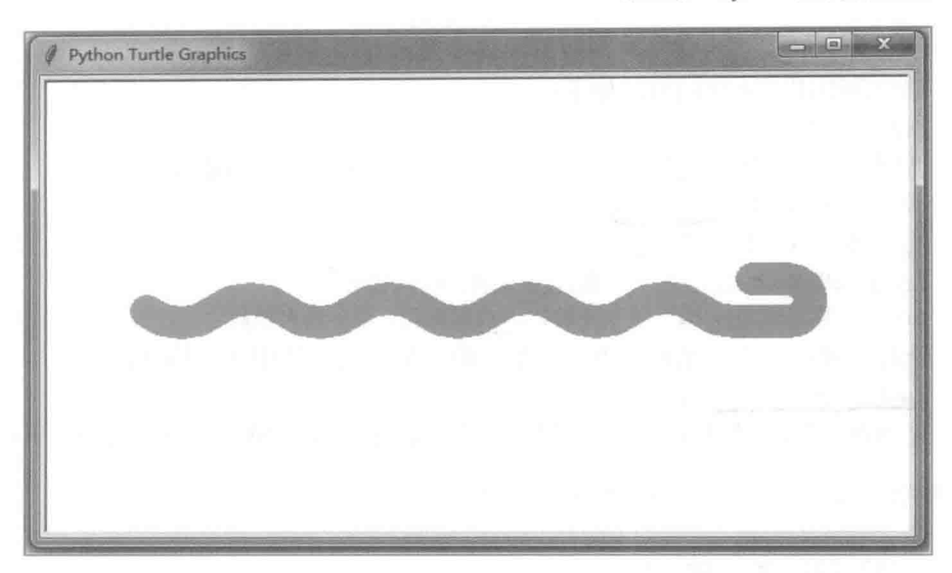


图 2.3　Python 蟒蛇绘制的输出效果

实例代码 2.1 与实例代码 1.1 有两个显著的不同。第一，这个程序没有使用显式的用户输入输出，即没有 input()函数和 print()函数；第二，这个程序绝大多数代码行都是<a>.<b>()形式，代码行中没有赋值语句。

<a>.<b>()是 Python 编程的一种典型表达形式，它可以表示调用一个对象[1]<a>的方法<b>()，也可以表示调用一个函数库<a>中的函数<b>()。

实例代码 2.1 使用了用于绘制图形的 turtle 库，并在第 2 行代码中通过保留字 import 引用这个函数库。

```
2    import turtle
```

实例代码 2.1 的第 3 行到第 16 行调用了 turtle 库中若干函数来绘制 Python 蟒蛇，所有被调用的函数都使用了<a>.<b>()形式。这种通过使用函数库并利用库中函数进行编程的方法是 Python 语言最重要的特点，称为“模块编程”。8.5 节将详细介绍 Python 模块编程思想以及本书所提出的面向“计算生态”的教学理念。

拓展： 面向对象编程

面向对象编程（Object-Oriented Programming，OOP）是一种基于对象（Object）的编程范式。对象是事物的一种抽象，它是一个实体，包含属性和方法两部分。属性是对象中的变量，方法是对象能够完成的操作。

假设对象是 O，则 O.a 表示对象 O 的属性 a，O.b()表示对象 O 的操作 b()，其中 a 是一个变量值，b()是一个函数。例如，一辆汽车可以作为一个对象，标记为 C，汽车的颜色是汽车的属性，表示为 C.color，前进是汽车的一个动作，相当于一个功能，因此前进是对象 C 的方法，表示为 O.forward()。

[1] 本书不讲解面向对象编程，因此，读者只需了解与对象有关的基本概念即可。

使用 import 引用函数库有两种方式，但对函数的使用方式略有不同。

第一种引用函数库的方法如下：

```
import <库名>
```

此时，程序可以调用库名中的所有函数，使用库中函数的格式如下：

```
<库名>.<函数名>(<函数参数>)
```

第二种引用函数库的方法如下：

```
from  <库名>  import <函数名,函数名,…,函数名>
from  <库名>  import *    #其中，*是通配符，表示所有函数
```

此时，调用该库的函数时不再需要使用库名，直接使用如下格式：

```
<函数名>(<函数参数>)
```

采用第二种库引用方式修改实例代码 2.2 完成 Python 蟒蛇绘制，代码如下：

源代码 2-5：Python 蟒蛇绘制程序



实例代码 2.2　　　　e2.2DrawPython.py

```
1   #e2.2DrawPython.py
2   from turtle import *
3   setup(650, 350, 200, 200)
4   penup()
5   fd(-250)
6   pendown()
7   pensize(25)
8   pencolor("purple")
9   seth(-40)
10  for i in range(4):
11      circle(40, 80)
12      circle(-40, 80)
13  circle(40, 80/2)
14  fd(40)
15  circle(16, 180)
16  fd(40 * 2/3)
```

实例代码 2.2 与实例代码 2.1 运行结果相同，所不同的是调用 turtle 库中函数时不再采用<a>.<b>()方式，而直接使用函数名。由于“Python 蟒蛇绘制”程序只用了 turtle 库中的 setup()、penup()、fd()、pendown()、pensize()、pencolor()、seth()、circle()等 8 个函数，第 2 行的 import 语句也可以写成如下形式：

```
2   from turtle import setup, penup, fd, pendown
3   from turtle import pensize, pencolor, seth, circle
```

两种函数库引用方式各有优点。第一种采用<a>.<b>()方式调用库中函数，能够显式标明函数来源，在引用较多库时代码可读性更好。第二种利用保留字直接引用库中函数，可以使代码更简洁，在类似实例代码 2.2 这种只引用一个库的情况下，效果更好。

需要注意的是，第一种引用方式，Python 解释器将<a>.<b>整体作为函数名。当采用第二种方式时，Python 解释器将<b>作为函数名。这可能产生一种情况，假设用户已经定义了一个函数<b>，库中的函数名<b>将会与用户自定义的函数名冲突。由于 Python 程序要求函数命名唯一，所以，当函数名冲突时 Python 解释器会以最近的函数定义为准。为了避免可能的命名冲突，对于初学者，建议采用第一种库引用方式，使用<a>.<b>()方式调用库函数。

思考与练习

2.8　请修改实例代码 2.1 中第 8 行代码，将"purple"变为"violet"，观察程序运行结果的变化。

2.9　请修改实例代码 2.1 中第 10 行代码，将 range(4)变为 range(5)，观察程序运行结果的变化。

2.10　请修改实例代码 2.1 中第 4 行和第 6 行代码，在两行的最前面增加注释符号，即将这两行变成注释语句，观察程序运行结果的变化。

2.4　turtle 库语法元素分析

要点： 结合“Python 蟒蛇绘制”实例，分析 turtle 库语法元素，包括绘图坐标体系、画笔控制函数和形状绘制函数等。

图片资料 2-2：Python 快速参考之 turtle 库



Python 的 turtle 库是一个直观有趣的图形绘制函数库。turtle（海龟）图形绘制的概念诞生于 1969 年，并成功应用于 LOGO 编程语言。由于 turtle 图形绘制概念十分直观且非常流行，Python 接受了这个概念，形成了一个 Python 的 turtle 库，并成为标准库之一。9.4 节将全面介绍 turtle 库的使用，本书末尾将给出完整的快速参考。为了介绍 Python 模块编程思想并解释“Python 蟒蛇绘制”程序，本节结合实例代码 2.1 介绍 turtle 库中部分函数的使用，这些函数将同时用于后续章节的部分实例中。

2.4.1　绘图坐标体系

turtle 库绘制图形有一个基本框架：一个小海龟在坐标系中爬行，其爬行轨迹形成了绘制图形。对于小海龟来说，有“前进”、“后退”、“旋转”等爬行行为，对坐标系的探索也通过“前进方向”、“后退方向”、“左侧方向”和“右侧方向”等小海龟自身角度方位来完成。刚开始绘制时，小海龟位于画布正中央，此处坐标为(0, 0)，行进方向为水平右方。例如，用如下代码绘制如图 2.4 所示的图坐标体系。

```
3    turtle.setup(650, 350, 200, 200)
```

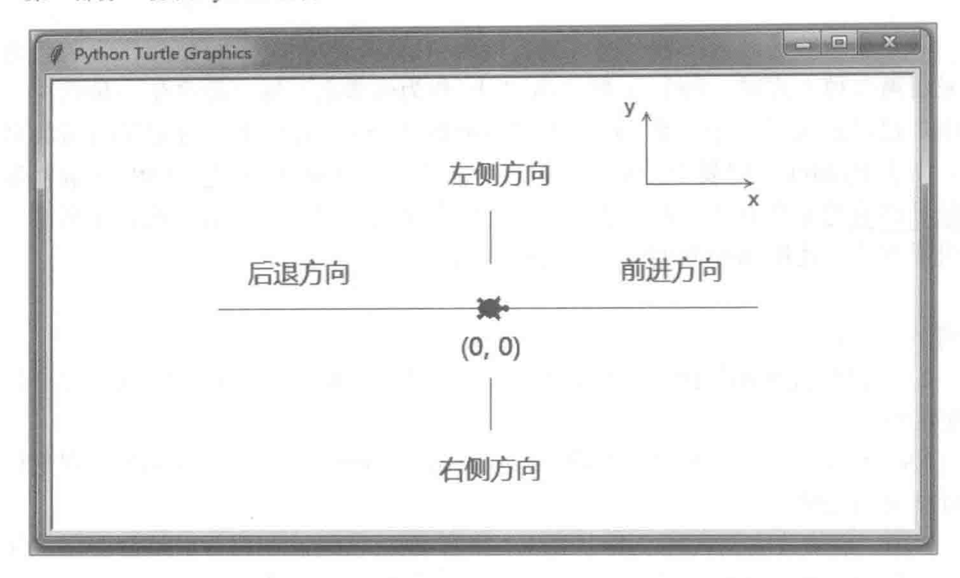


图 2.4 Python turtle 库绘图坐标体系

实例代码 2.1 第 3 行使用了 turtle.setup()函数，该函数各参数的关系如图 2.5 所示，其具体定义如下。

```
turtle.setup(width, height, startx, starty)
```

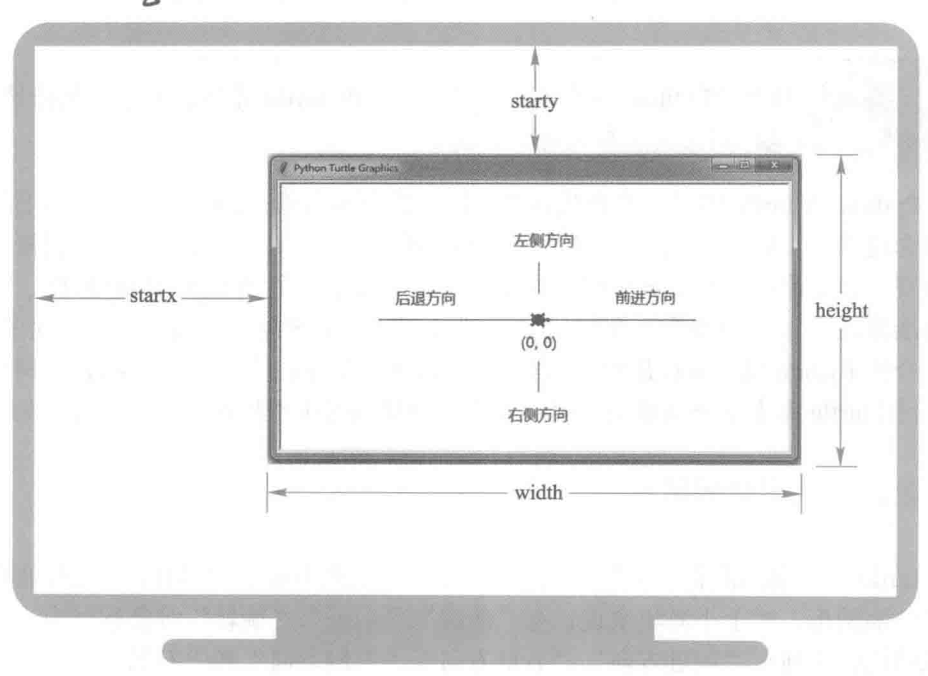


图 2.5 turtle.setup()函数 4 个参数的含义

作用：设置主窗体的大小和位置。

参数如下。

width：窗口宽度，如果值是整数，表示像素值；如果值是小数，表示窗口宽度

与屏幕的比例。

height：窗口高度，如果值是整数，表示像素值；如果值是小数，表示窗口高度与屏幕的比例。

startx：窗口左侧与屏幕左侧的像素距离，如果值是 None，窗口位于屏幕水平中央。

starty：窗口顶部与屏幕顶部的像素距离，如果值是 None，窗口位于屏幕垂直中央。

2.4.2　画笔控制函数

1．turtle.penup()和 turtle.pendown()函数

```
4    turtle.penup()
6    turtle.pendown()       （实例代码 2.1 中第 4、6 行）
```

turtle 中的画笔（即小海龟）可以通过一组函数来控制，实例代码 2.1 中第 4 行的 turtle.penup()函数和第 6 行的 turtle.pendown()函数是一组，它们分别表示抬起画笔和落下画笔，函数定义如下：

turtle.penup()

别名

turtle.pu(), turtle.up()

作用：抬起画笔，之后移动画笔不绘制形状。

参数：无。

turtle.pendown()

别名

turtle.pd(), turtle.down()

作用：落下画笔，之后移动画笔将绘制形状。

参数：无。

2．turtle.pensize()函数

```
7    turtle.pensize(25)          （实例代码 2.1 中第 7 行）
```

turtle.pensize()函数用来设置画笔尺寸，函数定义如下：

turtle.pensize(width)

别名

turtle.width()

作用：设置画笔宽度，当无参数输入时返回当前画笔宽度。

参数如下。

width：设置的画笔线条宽度，如果为 None 或者为空，则函数返回当前画笔宽度。

3．turtle.pencolor()函数

```
8    turtle.pencolor("purple")          （实例代码 2.1 中第 8 行）
```

turtle.pencolor()函数给画笔设置颜色，实例代码 2.1 中将画笔设为“紫色”，函数定义如下：

```
turtle.pencolor(colorstring)
```

或

```
turtle.pencolor((r,g,b))
```

作用：设置画笔颜色，当无参数输入时返回当前画笔颜色。

参数如下。

```
colorstring ：表示颜色的字符串，例如，"purple"、"red"、"blue"等。
(r,g,b) ：颜色对应的 RGB 数值，例如，(51, 204, 140)。
```

很多 RGB 颜色都有固定的英文名字，这些英文名字可以作为 colorstring 输入到 turtle.pencolor()函数中，也可以采用(r, g, b)形式直接输入颜色值。几种典型的 RGB 颜色如表 2.2 所示。

表 2.2 部分典型 RGB 颜色对照表

英文名称	RGB	十六进制	中文名称
white	255 255 255	#FFFFFF	白色
black	0 0 0	#000000	黑色
grey	190 190 190	#BEBEBE	灰色
darkgreen	0 100 0	#006400	深绿色
gold	255 215 0	#FFD700	金色
violet	238 130 238	#EE82EE	紫罗兰
purple	160 32 240	#A020F0	紫色

拓展： RGB 颜色

RGB 颜色是计算机系统最常用的颜色体系之一，它采用 R（红色）、G（绿色）、B（蓝色）3 种基本颜色及它们的叠加组成各式各样的颜色，构成颜色体系。RGB 颜色诞生于 19 世纪中期、计算机产生之前，理论表明，RGB 颜色能够形成人眼感知的所有颜色。

具体来说，RGB 颜色采用（r, g, b）表示，其中，每个颜色采用 8 bit 表示，取值范围是[0, 255]。因此，RGB 颜色一共可以表示 256^3（16 M，约 1 678 万）种颜色。

2.4.3 形状绘制函数

1. turtle.fd()函数

```
5    turtle.fd(-250)
14   turtle.fd(40)            （实例代码 2.1 中第 5、14、16 行）
16   turtle.fd(40 * 2/3)
```

turtle 通过一组函数控制画笔的行进动作，进而绘制形状。turtle.fd()函数最常用来控制画笔向当前行进方向前进一个距离，函数定义如下：

turtle.fd(distance)

别名

turtle.forward(distance)

作用：向小海龟当前行进方向前进 distance 距离。

参数如下。

distance：行进距离的像素值，当值为负数时，表示向相反方向前进。

实例代码 2.1 中第 5、14、16 行分别表示向画笔当前前进方向或反方向行进一段距离，进而绘制一条直线。第 5 行代码由于配合了第 4 行抬起画笔函数，因此，它将不绘制一条直线，而是将画笔移动到某个位置。

2．turtle.seth()函数

```
9    turtle.seth(-40)
```

turtle.seth()函数用来改变画笔绘制方向，函数定义如下：

turtle.seth(to_angle)

别名

turtle.setheading(to_angle)

作用：设置小海龟当前行进方向为 to_angle，该角度是绝对方向角度值。

参数如下。

to_angle：角度的整数值。

如图 2.6 所示是 turtle 库的角度坐标体系，供 turtle.seth()等函数使用。需要注意

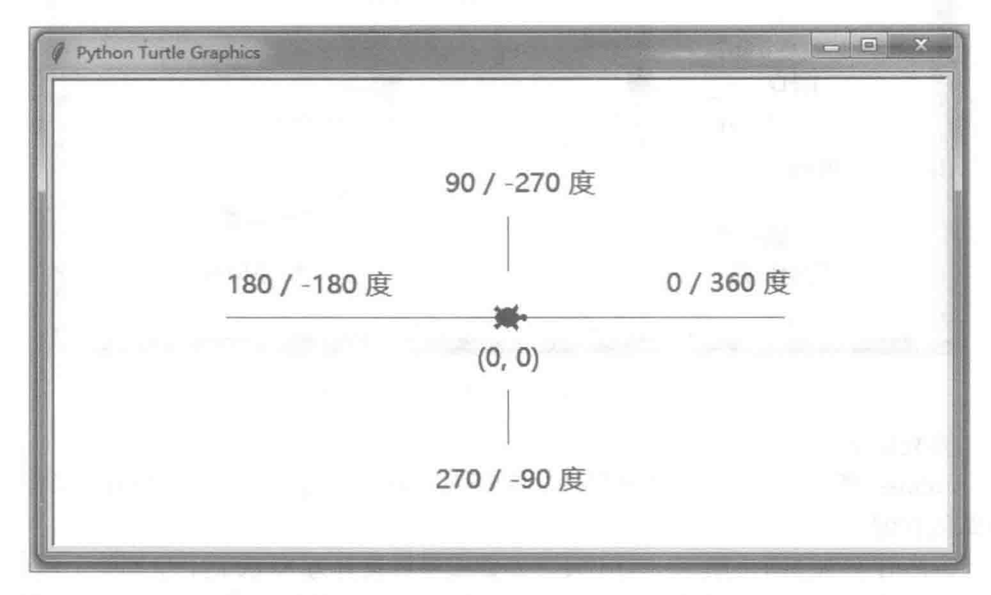


图 2.6　turtle 库的角度坐标体系

的是，turtle 库的角度坐标体系以正东向为绝对 0 度，这也是小海龟初始爬行方向，正西向为绝对 180 度，这个方向坐标体系是方向的绝对方向体系，与小海龟爬向当前方向无关。因此，可以利用这个绝对坐标体系随时更改小海龟的前进方向。

实例代码 2.1 中第 9 行将小海龟的前进方向设定为-40 度，即 320 度，之后小海龟向这个方向前进。

3．for 循环语句和 turtle.circle()函数

```
10  for i in range(4):
11      turtle.circle(40, 80)
12      turtle.circle(-40, 80)     （实例代码 2.1 中第 10～15 行）
13  turtle.circle(40, 80/2)
15  turtle.circle(16, 180)
```

由于存在缩进，实例代码 2.1 中第 10、11、12 行是一个由保留字 for 引导的整体，这是另一种循环结构，称为“遍历循环”。for 语句的循环格式如下：

```
for i in range(<循环次数>):
    <语句块 1>
```

这里请读者记住这种结构，4.4 节将深入介绍这种循环结构的使用。实例代码 2.1 中第 10 行的 for 循环表示第 11、12 行代码连续执行 4 次。

turtle.circle()函数用来绘制一个弧形，函数定义如下（参数含义如图 2.7 所示）：

turtle.circle(radius, extent=None)

作用：根据半径 radius 绘制 extent 角度的弧形，绘制模式如图 2.7 所示。

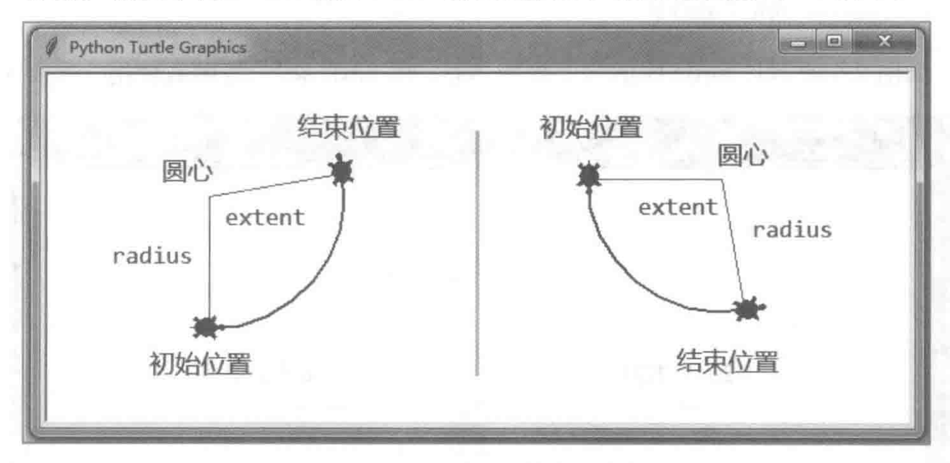


图 2.7 turtle.circle()函数的参数含义

参数如下。

raduis：弧形半径，当值为正数时，半径在小海龟左侧，当值为负数时，半径在小海龟右侧。

extent：绘制弧形的角度，当不设置参数或参数设置为 None 时，绘制整个圆形。

实例代码 2.1 中各函数包括角度值、半径值等参数是根据绘制内容的样式调整

确定的，在了解各函数使用含义的基础上，读者可以修改各函数参数值，观察蟒蛇绘制的效果。

2.4.4　函数的封装

实例代码 2.1 的程序功能可以分成两类：绘制图形前对画笔的设置，包括颜色、尺寸、初始位置等，以及绘制 Python 蟒蛇的功能。由于蟒蛇绘制的功能相对独立，可以用函数来封装，实例代码 2.3 给出了带有函数定义的程序。其中，第 3～11 行通过保留字 def 定义了 drawSnake()函数，将蟒蛇绘制这个独立功能封装起来。

源代码 2-6：函数封装的 Python 蟒蛇绘制程序



实例代码 2.3　　e2.3DrawPython.py

```
1   #e2.3DrawPython.py
2   import turtle
3   def drawSnake(radius, angle, length):
4       turtle.seth(-40)
5       for i in range(length):
6           turtle.circle(radius, angle)
7           turtle.circle(-radius, angle)
8       turtle.circle(radius, angle/2)
9       turtle.fd(40)
10      turtle.circle(16, 180)
11      turtle.fd(40* 2/3)
12  turtle.setup(650, 350, 200, 200)
13  turtle.penup()
14  turtle.fd(-250)
15  turtle.pendown()
16  turtle.pensize(25)
17  turtle.pencolor("purple")
18  drawSnake(40, 80, 4)
19  turtle.done()
```

程序练习 2-2：半小时学 turtle



通过保留字 def 定义的函数是自定义函数。自定义函数与 turtle 库提供的函数不同，它们是用户自己定义实现的。第 5 章将详细介绍有关函数的功能和使用。

阶段测试 2-2：turtle 入门小测验



思考与练习

2.11　请使用 turtle 库的 turtle.fd()函数绘制一条直线。
2.12　请使用 turtle 库中 turtle.circle()函数绘制一个完整的圆。
2.13　请使用 turtle 库函数绘制一个包含 9 个同心圆的靶盘。

本章小结

本章从解决实际问题入手，以简单的温度转换程序为例，逐行逐句地分析了 Python 语言的基本元素。通过讲解 Python 蟒蛇绘制实例，介绍 Python 语言函数库 turtle 及其基本用法。

程序练习 2-3：
章节程序练习题



程序练习题

2.1 实例 1 的修改。改造实例代码 1.1，采用 eval(input(<提示内容>))替换现有输入部分，并使输出的温度值为整数。

2.2 汇率兑换程序。按照温度转换程序的设计思路，按照 1 美元=6 人民币汇率编写一个美元和人民币的双向兑换程序。

2.3 实例 2 的修改。改造实例代码 2.1，绘制一条彩色蟒蛇，即在绘制 Python 蟒蛇的每个小段时，画笔的绘制颜色会发生变化。

提示：将画笔颜色控制函数放到蟒蛇绘制函数附近。

2.4 等边三角形的绘制。使用 turtle 库中的 turtle.fd()函数和 turtle.seth()函数绘制一个等边三角形，效果如图 2.8 所示。

2.5 叠加等边三角形的绘制。使用 turtle 库中的 turtle.fd()函数和 turtle.seth()函数绘制一个叠加等边三角形，效果如图 2.9 所示。

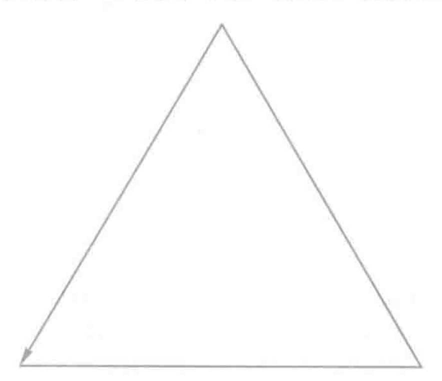


图 2.8 等边三角形的绘制效果

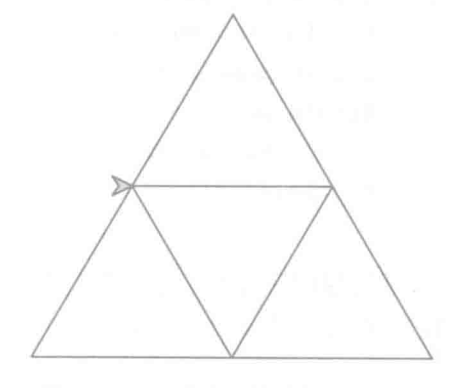


图 2.9 叠加等边三角形的绘制效果

2.6 无角正方形的绘制。利用 turtle 库函数绘制一个没有角的正方形，效果如图 2.10 所示。

2.7 六角形的绘制。利用 turtle 库绘制一个六角形，效果如图 2.11 所示。

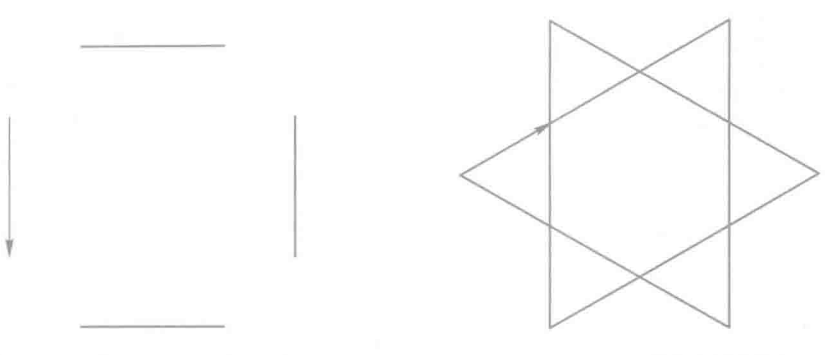


图 2.10　无角正方形的绘制效果　　　　图 2.11　六角形的绘制效果

2.8　正方形螺旋线的绘制。利用 turtle 库绘制一个正方形螺旋线，效果如图 2.12 所示。

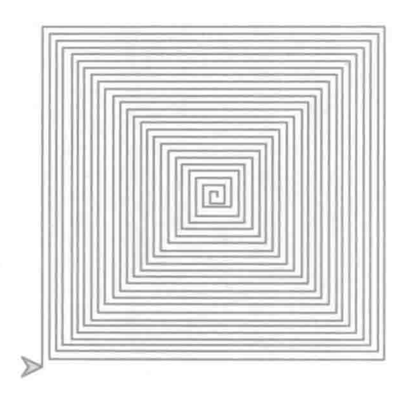


图 2.12　正方形螺旋线的绘制效果

2.9　自定义 Python 蟒蛇绘制。根据实例 2 的设计思想，结合读者喜好，绘制一条区别于实例 2 的 Python 蟒蛇。

第二部分　深入 Python 语言

MOOC 课程：中国大学 MOOC"Python 语言程序设计"课程



本部分主要讲解 Python 语言的基本语法，建立对 Python 语言常用语法体系的基本理解，使读者掌握利用 Python 语言编写程序的能力，初步掌握利用程序解决计算问题的方法。本部分的学习目标是编写 20 行左右的 Python 程序。

本部分包括 5 章内容（第 3 ~ 7 章），分别如下：

第 3 章　基本数据类型

第 4 章　程序的控制结构

第 5 章　函数和代码复用

第 6 章　组合数据类型

第 7 章　文件和数据格式化

第 3 章主要讲解 Python 语言的基本数据类型，包括整数、浮点类、复数、字符串等类型的概念和使用，同时，介绍标准库 math 的使用。

第 4 章主要讲解 Python 语言的指令流控制结构，包括顺序结构、分支结构、循环结构、异常处理结构等，同时，介绍标准库 random 的使用。

第 5 章主要讲解函数的概念，包括函数的基本使用、函数的参数传递、代码复用、基于函数的模块化设计、递归等，同时，介绍标准库 datetime 的使用。

第 6 章主要讲解 Python 语言的组合数据类型，包括元组、集合、列表、字典等类型的概念和使用，同时，介绍第三方中文分词库 jieba 的使用。

第 7 章主要讲解文件和数据格式化，包括文件的使用以及一二维和高维数据组织和格式化方法，介绍第三方图像处理库 PIL 和标准库 JSON 的使用。

本部分按照数据表示、程序结构和抽象交互 3 个层次依次讲解 Python 语言语法，其中，数据表示对应第 3 章和第 6 章；程序结构对应第 4 章；抽象交互对应第 5 章和第 7 章。建议读者按照章节顺序阅读和学习本部分内容。

第 3 章　基本数据类型

电子教案 3-1：
基本数据类型



只有两种编程语言：一种是经常被骂的，一种是没人使用的。

There are only two kinds of programming languages: those people always bitch about and those nobody uses.

——本贾尼·斯特劳斯特卢普（Bjarne Stroustrup）
C++语言之父

学习目标

（1）掌握 3 种数字类型的概念和使用。
（2）了解 3 种数字类型在计算机中的表示方法。
（3）运用 Python 的标准数学库进行数值计算。
（4）掌握字符串类型的概念和使用。
（5）掌握字符串类型的格式化操作方法和应用。

1951 年，毛泽东主席题词“好好学习、天天向上”，成为激励一代代中国人奋发图强的经典短语。可是在实际操作中难免会让人疑惑，“好好学习”究竟能好到什么程度？“天天向上”难道要全年 365 天完全无休？本章将采用编程思想，充分考虑实际努力因素和时间累积程度，使用 Python 来演算天天向上的力量。

“好好学习、天天向上”，它的力量十分惊人！

3.1 数 字 类 型

要点：Python 语言提供整数、浮点数、复数 3 种数字类型。

3.1.1 数字类型概述

数字是自然界计数活动的抽象，更是数学运算和推理表示的基础。计算机对数字的识别和处理有两个基本要求：确定性和高效性。

确定性指程序能够正确且无歧义地解读数据所代表的类型含义。例如，输入 1010，计算机需要明确地知道这个输入是可以用来进行数学计算的数字 1010，还是类似房间门牌号一样的字符串"1010"，这两者用处不同、操作不同且在计算机内部存储方式不同。即便 1010 是数字，还需要进一步明确这个数字是十进制、二进制还是其他进制类型。

高效性指程序能够为数字运算提供较高的计算速度，同时具备较少的存储空间代价。整数和带有小数的数字分别由计算机中央处理器中不同的硬件逻辑操作，对于相同类型操作，如整数加法和小数加法，前者比后者的速度一般快 5～20 倍。为了尽可能提高运行速度，需要区分不同运行速度的不同数字类型。

表示数字或数值的数据类型称为数字类型，Python 语言提供 3 种数字类型：整数、浮点数和复数，分别对应数学中的整数、实数和复数。1010 表示一个整数，"1010" 表示一个字符串。

3.1.2 整数类型

整数类型与数学中整数的概念一致，下面是整数类型的例子：

1010, 99, -217, 0x9a, -0x89

整数类型共有 4 种进制表示：十进制、二进制、八进制和十六进制。默认情况，整数采用十进制，其他进制需要增加引导符号，如表 3.1 所示。二进制数以 0b 引导，八进制数以 0o 引导，十六进制数以 0x 引导，大小写字母均可使用。

表 3.1 整数类型的 4 种进制表示

进制种类	引导符号	描述
十进制	无	默认情况，例如，1010,-425
二进制	0b 或 0B	由字符 0 和 1 组成，例如，0b101，0B101
八进制	0o 或 0O	由字符 0 到 7 组成，例如，0o711，0O711
十六进制	0x 或 0X	由字符 0 到 9、a 到 f、A 到 F 组成，例如，0xABC

整数类型理论上的取值范围是[-∞,∞]，实际上的取值范围受限于运行 Python 程序的计算机内存大小。除极大数的运算外，一般认为整数类型没有取值范围限制。

pow(x,y)函数是 Python 语言的一个内置函数[1]，用来计算 x^y。这里，用 pow()函数测试一下整数类型的取值范围，例如：

```
>>>pow(2,100)
1267650600228229401496703205376
>>>pow(2,500)
3273390607896141870013189696827599152216642046043064789483291368096
1337964046745548832700923259041571508866841275600710092172565458853
93053328527589376
```

pow()函数还可以嵌套使用，例如：

```
>>>pow(2,pow(2,15))
(此处省略输出结果)
```

上述程序可以在一般计算机上运行并输出运算结果，pow(2,pow(2,15))的结果是一个 9 865 位的整数。增加程序中第二个 pow()函数的幂会获得更大的输出结果，例如，将数值 15 改为 16，这会消耗更长的计算时间并占用更多的计算机内存。

3.1.3　浮点数类型

浮点数类型与数学中实数的概念一致，表示带有小数的数值。Python 语言要求所有浮点数必须带有小数部分，小数部分可以是 0，这种设计可以区分浮点数和整数类型。浮点数有两种表示方法：十进制表示和科学计数法表示。下面是浮点数类型的例子：

```
0.0, -77., -2.17, 3.1416, 96e4, 4.3e-3, 9.6E5
```

科学计数法使用字母 e 或 E 作为幂的符号，以 10 为基数，含义如下：

$<a>e<b> = a*10^b$

上例中 4.3e-3 值为 0.0043；9.6E5 也可以表示为 9.6E+5，其值为 960 000.0。

浮点数类型与整数类型由计算机的不同硬件单元执行，处理方法不同，需要注意的是，尽管浮点数 0.0 与整数 0 值相同，但它们在计算机内部表示不同。

Python 浮点数的数值范围和小数精度受不同计算机系统的限制，sys.float_info 详细列出了 Python 解释器所运行系统的浮点数各项参数，例如：

```
>>>import sys
>>>sys.float_info
sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_
exp=308, min=2.2250738585072014e-308, min_exp=-1021, min_10_
exp=-307, dig=15, mant_dig=53, epsilon=2.220446049250313e-16, radix=2,
```

[1] 内置函数指 Python 语言解释器包含的函数，这些函数可以直接使用。5.8 节给出了 Python 语言完整的内置函数列表。

```
rounds=1)
>>>sys.float_info.max
1.7976931348623157e+308
```

上述输出给出浮点数类型所能表示的最大值（max）、最小值（min），科学计数法表示下最大值的幂（max_10_exp）、最小值的幂（min_10_exp），基数（radix）为 2 时最大值的幂（max_exp）、最小值的幂（min_exp），科学计数法表示中系数（<a>）的最大精度（mant_dig），计算机所能分辨的两个相邻浮点数的最小差值（epsilon），能准确计算的浮点数最大个数（dig）。

浮点数类型直接表示或科学计数法表示中的系数（<a>）最长可输出 16 个数字，浮点数运算结果中最长可输出 17 个数字，然而，根据 sys.float_info 结果，计算机只能够提供 15 个数字（dig）的准确性，最后一位由计算机根据二进制计算结果确定，存在误差，例如：

```
>>>3.1415926535897924
3.1415926535897922
>>>987654321123456.789
987654321123456.8
```

浮点数在超过 15 位数字计算中产生的误差与计算机内部采用二进制运算有关，使用浮点数无法进行极高精度的数学运算。

由于 Python 语言能够支持无限制且准确的整数计算，因此，如果希望获得精度更高的计算结果，往往采用整数而不直接采用浮点数。例如，计算如下两个数的乘法值，它们的长度只有 10 个数字，其中：

a=3.141592653，b=1.234567898

可以直接采用浮点数运算，也可以同时把它们的小数点去掉，当作整数运算，结果如下：

```
>>>3.141592653*1.234567898
3.8785094379864535
>>>3141592653*1234567898
3878509437986453394
```

其中，浮点数运算输出 17 个数字长度的结果，然而，只有前 15 个数字是确定正确的。整数运算能够输出完全准确的运算结果。使用整数表达浮点数的方法是高精确度运算的基本方法之一。

拓展： 高精度浮点运算类型

Python 通过标准库 decimal 提供了一个更精确的数字类型 Decimal，这个类型利用上文所介绍的整数运算方法提供高精度浮点数运算，并可以使用 getcontext().prec 参数自定义浮点数精度的位数，例如：

```
>>>import decimal
>>>a=decimal.Decimal('3.141592653')
```

```
>>>b=decimal.Decimal('1.234567898')
>>>decimal.getcontext().prec=20
>>>a*b
Decimal(' 3.8785094379864533394')
```

需要注意的是，在使用 decimal 库时，Decimal('数字')是高精确度数字的基本表示形式，需要使用单引号，例如，decimal.Decimal('3.141592653')。

简单地说，浮点数类型的取值范围在$[2^{-1\,023}, 2^{1\,023}]$，即$[-2.225\times10^{308}, 1.797\times10^{308}]$之间，运算精度为 2.220×10^{-16}，即浮点数运算误差仅为 0.000 000 000 000 000 2。对于高精度科学计算外的绝大部分运算来说，浮点数类型足够“可靠”，一般认为浮点数类型没有范围限制，运算结果准确。

3.1.4　复数类型

复数类型表示数学中的复数。很久以前，数学界被求解如下等式难住了：

$$x^2=-1$$

这是因为任何实数都不是上述等式的解。直到 18 世纪，数学家发明了“虚数单位”，记为 j，并规定 $j=\sqrt{-1}$。围绕这个特殊数字出现了新的数学分支，产生了“复数”。对于一个实数 n，根据上述定义，$n\times j\times j$ 的值是$-n$，图 3.1 给出了对虚数单位 j 的表示，如果将实数看成一个数轴，虚数看成与实数垂直的正交数轴，j 表示“逆时针旋转 90°”，或者，“逆时针旋转 π/4”。

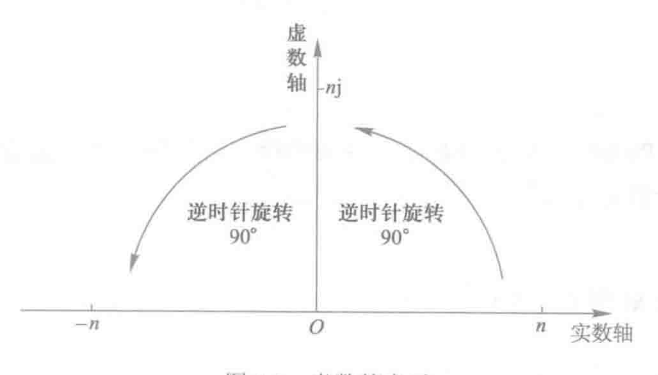


图 3.1　虚数的表示

复数可以看作是二元有序实数对(a, b)，表示为 $a + b$j，其中，a 是实数部分，简称实部，b 是虚数部分，简称虚部。根据图 3.1，复数是实数在二维平面空间旋转的一种表示。

Python 语言中，复数的虚数部分通过后缀“J”或“j”来表示，例如：

12.3+4j　　　−5.6+7j　　　1.23e−4+5.67e+89j

复数类型中实数部分和虚数部分的数值都是浮点类型。对于复数 z，可以用 z.real 和 z.imag 分别获得它的实数部分和虚数部分，例如：

```
>>>1.23e-4+5.67e+89j.real
0.000123
>>>1.23e-4+5.67e+89j.imag
5.67e+89
```

复数类型在科学计算中十分常见，基于复数的运算属于数学的复变函数分支，该分支有效支撑了众多科学和工程问题的数学表示和求解。Python 直接支持复数类型，为这类运算求解提供了便利。

思考与练习

3.1 既然浮点数可以表示所有整数数值，Python 语言为何要同时提供整数和浮点数两种数据类型？

3.2 Python 语言中整数 1010 的二进制、八进制和十六进制表示分别是什么？

二进制：__________、八进制：__________、十六进制：__________

3.3 Python 语言中-77.的科学计数法表示是什么？4.3e-3 的十进制表示是什么？

–77.的科学计数法表示：__________、4.3e-3 的十进制表示：__________

3.4 复数 2.3e+3-1.34e-3j 的实部和虚部分别是什么？采用什么方法提取一个复数的实部和虚部？

3.2 数字类型的操作

> **要点：** Python 解释器为数字类型提供数值运算操作符、数值运算函数、类型转换函数等操作方法。

3.2.1 内置的数值运算操作符

Python 提供了 9 个基本的数值运算操作符，如表 3.2 所示。这些操作符由 Python 解释器直接提供，不需要引用标准或第三方函数库，也叫做内置操作符。

表 3.2 内置的数值运算操作符（共 9 个）

操 作 符	描 述
x + y	x 与 y 之和
x - y	x 与 y 之差
x * y	x 与 y 之积
x / y	x 与 y 之商
x // y	x 与 y 之整数商，即不大于 x 与 y 之商的最大整数

续表

操 作 符	描 述
x % y	x 与 y 之商的余数，也称为模运算
–x	x 的负值，即 x*(-1)
+x	x 本身
x**y	x 的 y 次幂，即 x^y

这 9 个操作符与数学习惯一致，运算结果也符合数学意义。操作符运算的结果可能改变数字类型，3 种数字类型之间存在一种逐渐扩展的关系，具体如下：

整数 -> 浮点数 -> 复数

这是因为整数可以看成是浮点数没有小数的情况，浮点数可以看成是复数虚部为 0 的情况。基于上述扩展关系，数字类型之间相互运算所生成的结果是“更宽”的类型，基本规则如下。

（1）整数之间运算，如果数学意义上的结果是小数，结果是浮点数。

（2）整数之间运算，如果数学意义上的结果是整数，结果是整数。

（3）整数和浮点数混合运算，输出结果是浮点数。

（4）整数或浮点数与复数运算，输出结果是复数。

```
>>>100/3
33.333333333333336
>>>100//3
33
>>>123 + 4.0
127.0
>>>10.0 - 1 + 2j  # 等价于(10.0 - 1)+ 2j
(9+2j)
```

表 3.2 中所有二元数学操作符（+、-、*、/、//、%、**）都有与之对应的增强赋值操作符（+=、-=、*=、/=、//=、%=、**=）。如果用 op 表示这些二元数学操作符，则下面两个赋值操作等价，注意，op 和二元操作符之间没有空格：

```
x op= y
```

等价于

```
x = x op y
```

增强赋值操作符获得的结果写入变量 x 中，简化了代码表达，例如：

```
>>>x=3.141592653
>>>x**=3  #与 x = x**3 等价
>>>x
31.006276662836743
```

3.2.2 内置的数值运算函数

Python 解释器提供了一些内置函数，5.8 节将给出 Python 全部内置函数列表，

在这些内置函数之中，有 6 个函数与数值运算相关，如表 3.3 所示。





表 3.3 内置的数值运算函数（共 6 个）

函　　数	描　　述
abs(x)	x 的绝对值
divmod(x, y)	(x//y, x%y)，输出为二元组形式（也称为元组类型）
pow(x, y[, z])	(x**y)%z，[..]表示该参数可以省略，即 pow(x,y)，它与 x**y 相同
round(x[, ndigits])	对 x 四舍五入，保留 ndigits 位小数。round(x)返回四舍五入的整数值
max($x_1, x_2,\ldots, x_n$)	$x_1, x_2,\ldots, x_n$ 的最大值，n 没有限定
min($x_1, x_2,\ldots, x_n$)	$x_1, x_2, \ldots, x_n$ 的最小值，n 没有限定

abs()可以计算复数的绝对值。复数的实部和虚部可以参照图 3.1 来理解，因此，复数的绝对值是二维坐标系中复数位置到坐标原点的长度。例如：

```
>>>abs(-3+4j)
5.0
```

pow()函数第三个参数 z 是可选的，使用该参数时，模运算与幂运算同时进行，速度很快。例如，求 3 的 3^{999} 次幂结果的最后 4 位。从 Python 语法角度，pow(3,pow(3,999))%10000（请不要在计算机中尝试该语句）和 pow(3, pow(3,999),10000)都能完成计算需求。但是，前者是先求幂运算结果再进行模运算，由于幂运算结果数值巨大，上述计算在一般计算机上无法完成；而第二条语句则在幂运算同时进行模运算，可以很快计算出结果。例如：

```
>>>pow(3, pow(3,999),10000) # 幂运算和模运算同时进行，速度快
4587
```

pow()函数第三个参数 z 的这个特点在运算加解密算法和科学计算中十分重要。

拓展： 模运算

模运算（%）在编程中十分常用，主要应用于具有周期规律的场景。例如，一个星期 7 天，用 day 代表日期，则 day%7 可以表示星期；对于一个整数 n，n%2 的取值是 0 或者 1，可以判断整数 n 的奇偶。本质上，整数的模运算 n%m 能够将整数 n 映射到[0, m–1]的区间中。

3.2.3 内置的数字类型转换函数

数值运算操作符可以隐式地转换输出结果的数字类型，例如，两个整数采用运算符“/”的除法将可能输出浮点数结果。此外，通过内置的数字类型转换函数可以显式地在数字类型之间进行转换，如表 3.4 所示。

浮点数类型转换为整数类型时，小数部分会被舍弃（不使用四舍五入），复数不能直接转换为其他数字类型，可以通过.real 和.imag 将复数的实部或虚部分别转换，例如：

表 3.4 内置的数字类型转换函数（共 3 个）

函 数	描 述
int(x)	将 x 转换为整数，x 可以是浮点数或字符串
float(x)	将 x 转换为浮点数，x 可以是整数或字符串
complex(re[, im])	生成一个复数，实部为 re，虚部为 im，re 可以是整数、浮点数或字符串，im 可以是整数或浮点数但不能为字符串

```
>>>int(10.99)
10
>>>complex(10.99)
(10.99+0j)
>>>float(10 + 99j) # 解释器会报 TypeError 错误，并给出该错误的基本描述
Traceback (most recent call last):
 File "<pyshell#77>", line 1, in <module>
   float(10 + 99j)
TypeError: can't convert complex to float
>>>float((10 + 99j).imag)
99.0
```

阶段测试 3-1：Python 数字类型小测试



程序练习 3-1：半小时学 Python 数字类型



思考与练习

3.5 思考各操作符的优先级，计算下列表达式。

（1）30-3**2+8//3**2*10

（2）3*4**2/8%5

（3）2**2**3

（4）(2.5+1.25j)*4j/2

3.6 请将下列数学表达式用 Python 程序写出来，并运算结果。

（1）$x = \frac{(2^4 + 7 - 3 \times 4)}{5}$

（2）$x = (1 + 3^2) \times (16 \bmod 7) / 7$

3.7 假设 x=1，x*=3+5**2 的运算结果是什么？

3.3 模块 1：math 库的使用

> 要点：Python 数学计算的标准函数库 math 共提供 4 个数学常数和 44 个函数。

3.3.1 math 库概述

利用函数库编程是 Python 语言最重要的特点，也是 Python 编程生态环境的意

义所在。本书不区分函数库（Library）和模块（Module），对于所有需要 import 使用的代码统称为函数库，这种利用函数库编程的方式称为“模块编程”。

从本节开始，本书将随各章节介绍一些常用的 Python 函数库。这些库分为 Python 环境中默认支持的函数库，以及第三方提供需要进行安装的函数库，其中默认支持的函数库也叫做标准函数库（Standard Library）或内置函数库。8.5 节将详细介绍 Python 函数库和模块编程思想。

图片资料 3-1：Python 快速参考之 math 库



math 库是 Python 提供的内置数学类函数库，因为复数类型常用于科学计算，一般计算并不常用，因此 math 库不支持复数类型，仅支持整数和浮点数运算。math 库一共提供了 4 个数学常数和 44 个函数。44 个函数共分为 4 类，包括 16 个数值表示函数、8 个幂对数函数、16 个三角对数函数和 4 个高等特殊函数。

math 库中函数数量较多，读者在学习过程中只需要逐个理解函数功能，记住个别常用函数即可。实际编程中，如果需要采用 math 库，可以随时查看本书附录提供的 math 库快速参考。

math 库中的函数不能直接使用，需要首先使用保留字 import 引用该库，引用方式如下。

第一种：

```
import math
```

对 math 库中函数采用 math.<b>()形式使用，例如：

```
>>>import math
>>>math.ceil(10.2)
11
```

第二种：

```
from math import <函数名>
```

对 math 库中函数可以直接采用<函数名>()形式使用，例如：

```
>>>from math import floor
>>>floor(10.2)
10
```

第二种方法的另一种形式是 from math import *。如果采用这种方式引入 math 库，math 库中所有函数可以采用<函数名>()形式直接使用。

math 库及后续所有函数库的引用都可以自由选取这两种方式实现，这与 2.3 节介绍的 turtle 库是一致的。

3.3.2 math 库解析

math 库包括 4 个数学常数，如表 3.5 所示；也包括 16 个数值表示函数，如表 3.6 所示。

表 3.5 math 库的数学常数（共 4 个）

常 数	数学表示	描 述
math.pi	π	圆周率，值为 3.141 592 653 589 793
math.e	e	自然对数，值为 2.718 281 828 459 045
math.inf	∞	正无穷大，负无穷大为-math.inf
math.nan		非浮点数标记，NaN（Not a Number）

表 3.6 math 库的数值表示函数（共 16 个）

函 数	数学表示	描 述
math.fabs(x)	$\|x\|$	返回 x 的绝对值
math.fmod(x, y)	$x\,\%\,y$	返回 x 与 y 的模
math.fsum([x,y,…])	$x+y+\cdots$	浮点数精确求和
math.ceil(x)	$\lceil x \rceil$	向上取整，返回不小于 x 的最小整数
math.floor(x)	$\lfloor x \rfloor$	向下取整，返回不大于 x 的最大整数
math.factorial(x)	$x!$	返回 x 的阶乘，如果 x 是小数或负数，返回 ValueError[1]
math.gcd(a, b)		返回 a 与 b 的最大公约数
math.frepx(x)	$x = \mathrm{m}\times 2\mathrm{e}$	返回(m, e)，当 x=0，返回(0.0, 0)
math.ldexp(x, i)	$x\times 2^{i}$	返回 $x\times 2^{i}$ 运算值，math.frep$x$(x)函数的反运算
math.modf(x)		返回 x 的小数和整数部分
math.trunc(x)		返回 x 的整数部分
math.copysign(x, y)	$\|x\|\times\|y\|/y$	用数值 y 的正负号替换数值 x 的正负号
math.isclose(a,b)		比较 a 和 b 的相似性，返回 True 或 False
math.isfinite(x)		当 x 为无穷大，返回 True；否则，返回 False
math.isinf(x)		当 x 为正数或负数无穷大，返回 True；否则，返回 False
math.isnan(x)		当 x 是 NaN，返回 True；否则，返回 False

math.fsum([x,y,…])函数在数学求和运算中十分有用，参考如下例子：

```
>>>0.1 + 0.2 + 0.3
0.6000000000000001
>>>import math
>>>math.fsum([0.1, 0.2, 0.3])
0.6
```

浮点数，如 0.1、0.2 和 0.3，在 Python 解释器内部表示时存在一个小数点后若干位的精度尾数，当浮点数进行运算时，这个精度尾数可能会影响输出结果。因此，在涉及浮点数运算及结果比较时，建议采用 math 库提供的函数，而不直接使用 Python 提供的运算符。

[1] ValueError 是 Python 编译器提供的一种异常，有关异常处理的内容详见 4.7 节。

math 包含 8 个幂对数函数，如表 3.7 所示。

表 3.7　math 库的幂对数函数（共 8 个）

函　　数	数学表示	描　　述
math.pow(x,y)	xy	返回 x 的 y 次幂
math.exp(x)	ex	返回 e 的 x 次幂，e 是自然对数
math.expml(x)	ex-1	返回 e 的 x 次幂减 1
math.sqrt(x)	$\sqrt{x}$	返回 x 的平方根
math.log(x[,base])	$\log_{base}x$	返回 x 的对数值，只输入 x 时，返回自然对数，即 $\ln x$
math.log1p(x)	$\ln(1+x)$	返回 1+x 的自然对数值
math.log2(x)	$\log x$	返回 x 的 2 对数值
math.log10(x)	$\log_{10}x$	返回 x 的 10 对数值

math 库没有提供直接支持 $\sqrt[y]{x}$ 运算的函数，但可以根据公式 $\sqrt[y]{x}=x^{\frac{1}{y}}$，采用 math.pow()函数求解，参考如下例子：

```
>>>math.pow(10, 1/3)
2.154434690031884
```

math 包含 16 个三角运算函数，如表 3.8 所示。

表 3.8　math 库的三角运算函数（共 16 个）

函　　数	数学表示	描　　述
math.degree(x)		角度 x 的弧度值转角度值
math.radians(x)		角度 x 的角度值转弧度值
math.hypot(x,y)	$\sqrt{x^2+y^2}$	返回(x,y)坐标到原点(0,0)的距离
math.sin(x)	$\sin x$	返回 x 的正弦函数值，x 是弧度值
math.cos(x)	$\cos x$	返回 x 的余弦函数值，x 是弧度值
math.tan(x)	$\tan x$	返回 x 的正切函数值，x 是弧度值
math.asin(x)	$\arcsin x$	返回 x 的反正弦函数值，x 是弧度值
math.acos(x)	$\arccos x$	返回 x 的反余弦函数值，x 是弧度值
math.atan(x)	$\arctan x$	返回 x 的反正切函数值，x 是弧度值
math.atan2(y,x)	$\arctan y/x$	返回 y/x 的反正切函数值，x 是弧度值
math.sinh(x)	$\sinh x$	返回 x 的双曲正弦函数值
math.cosh(x)	$\cosh x$	返回 x 的双曲余弦函数值
math.tanh(x)	$\tanh x$	返回 x 的双曲正切函数值
math.asinh(x)	$\text{arcsinh}\, x$	返回 x 的反双曲正弦函数值
math.acosh(x)	$\text{arccosh}\, x$	返回 x 的反双曲余弦函数值
math.atanh(x)	$\text{arctanh}\, x$	返回 x 的反双曲正切函数值

arctan 1 的值是 $\frac{\pi}{4}$，利用 math 库的 atan()函数得到π值如下：

```
>>>math.atan(1)*4
3.141592653589793
```

math 库的高等特殊函数共 4 个，如表 3.9 所示。

表 3.9　math 库的高等特殊函数（共 4 个）

函　　数	数 学 表 示	描　　述
math.erf(x)	$\frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}dt$	高斯误差函数，应用于概率论、统计学等领域
math.erfc(x)	$\frac{2}{\sqrt{\pi}}\int_x^\infty e^{-t^2}dt$	余补高斯误差函数，math.erfc(x)=1 - math.erf(x)
math.gamma(x)	$\int_0^\infty x^{t-1}e^{-x}dx$	伽玛（Gamma）函数，也叫欧拉第二积分函数
math.lgamma(x)	ln(gamma(x))	伽玛函数的自然对数

拓展： 伽玛函数

伽玛函数（Gamma Function，表示为 $\Gamma(x)$）不属于初等数学，然而，它却很有趣。以下 3 个公式都是伽玛函数的推广。

（1）x 为任意数，$\Gamma(x+1)=x\Gamma(x)$。

（2）当 x 为整数时，$\Gamma(x+1)=x\Gamma(x)=x(x-1)\Gamma(x-1)=\cdots=x!$。

（3）$\Gamma\left(\frac{1}{2}\right)=\sqrt{\pi}$。

因此，可以利用伽玛函数计算浮点数的“阶乘”，而 math.factorial()函数只能计算非负整数的阶乘，例如：

```
>>>math.factorial(10)
3628800
>>>math.gamma(11)
3628800.0
>>>math.gamma(-11)
3628800.0
>>>math.gamma(-10.2)
-9.184935416782052e-07
>>>math.factorial(-10.2)
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    math.factorial(-10.2)
ValueError: factorial() only accepts integral values
```

程序练习 3-2：半小时学 math 库



思考与练习

3.8 请利用 math 库运行下面语句，获得计算结果。

（1）math.sin(2*math.pi)　　（2）math.floor(–2.5)

（3）math.ceil(3.5+math.floor(–2.5))　　（4）round(math.fabs(–2.5))

（5）math.sqrt(math.pow(2,4))　　（6）math.log(math.e)

（7）math.gcd(12,9)　　（8）math.fmod(36,5)

3.9 请利用 math 库将 47°的角转换为弧度制，并将结果赋给一个变量。

阶段测试 3-2：math 库入门小测验



3.10 请利用 math 库将 $\frac{\pi}{7}$ 的弧度值转换为角度值，并将结果赋值给一个变量。

3.11 math 库有 44 个函数，Python 计算生态有超过 10 万个各类函数库，思考一下，该怎么学习这些函数库呢？

3.4 实例 3：天天向上的力量

要点： 这是一组测试“天天向上”力量的 Python 数学实例。

1951 年，毛泽东主席题词“好好学习、天天向上”，成为激励一代代中国人奋发图强的经典短语。那么“天天向上”的力量有多强大呢？这里用 Python 程序来演算一下吧。

【实例代码 3.1】天天向上。

一年 365 天，以第 1 天的能力值为基数，记为 1.0，当好好学习时能力值相比前一天提高 1‰，当没有学习时能力值相比前一天下降 1‰。每天努力和每天放任，一年下来的能力值相差多少呢？

根据题目，天天向上的力量是$(1+0.001)^{365}$，放任或向下的力量是$(1–0.001)^{365}$，代码如下：

源代码 3-1：天天向上的力量(1)



实例代码 3.1　　e3.1DayDayUp365.py

```
1  #e3.1DayDayUp365.py
2  import math
3  dayup = math.pow((1.0 + 0.001), 365)   # 提高 0.001
4  daydown = math.pow((1.0 - 0.001), 365) # 放任 0.001
5  print("向上: {:.2f}, 向下: {:.2f}.".format(dayup, daydown))
```

程序运行结果如下，每天努力 1‰，一年下来将提高 44%，好像不多？请继续分析。

```
>>>
向上: 1.44, 向下: 0.69.
```

【实例代码 3.2】天天向上。

一年 365 天，如果好好学习时能力值相比前一天提高 5‰，当放任时相比前一天下降 5‰，效果相差多少呢？

天天向上的力量是$(1+0.005)^{365}$，相反则是$(1-0.005)^{365}$，代码如下：



实例代码 3.2　　　　e3.2DayDayUp365.py

```
1  #e3.2DayDayUp365.py
2  import math
3  dayup = math.pow((1.0 + 0.005), 365)   # 提高 0.005
4  daydown = math.pow((1.0 - 0.005), 365) # 放任 0.005
5  print("向上: {:.2f}, 向下: {:.2f}.".format(dayup, daydown))
```

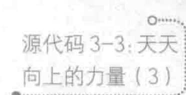
程序运行结果如下，每天努力 5‰，一年下来将提高 6 倍！这不容小觑了吧？

```
>>>
向上: 6.17, 向下: 0.16.
```

【实例代码 3.3】天天向上。

一年 365 天，如果好好学习时能力值相比前一天提高 1%，当放任时相比前一天下降 1%。效果相差多少呢？

此时，天天向上的力量是$(1+0.01)^{365}$，相反的力量是$(1-0.01)^{365}$。从 0.001、0.005 到 0.01，这个每天努力的因素根据需求的不同而不断变化，因此，新代码中采用 dayfactor 变量表示这个值。这种改变的好处是每次只需要修改这个变量值即可，而不需要修改后续与该变量相关位置的代码。代码如下：





实例代码 3.3　　　　e3.3DayDayUp365.py

```
1  #e3.3DayDayUp365.py
2  import math
3  dayfactor = 0.01
4  dayup = math.pow((1.0 + dayfactor), 365)    # 提高 dayfactor
5  daydown = math.pow((1.0 - dayfactor), 365)  # 放任 dayfactor
6  print("向上: {:.2f}, 向下: {:.2f}.".format(dayup, daydown))
```

程序运行结果如下，每天努力 1%，一年下来将提高 37 倍。这相当惊人吧！

```
>>>
向上: 37.78, 向下: 0.03.
```

【实例代码 3.4】天天向上。

一年 365 天，一周 5 个工作日，如果每个工作日都很努力，可以提高 1%，仅在周末放任一下，能力值下降 1%，效果如何呢？

当前水平值为 N，则工作日水平变化是 $N\times(1+0.01)$，非工作日是 $N\times(1-0.01)$。由于水平值并非每天都乘以相同系数，因此，这个程序采用循环方式实现。代码如下：

源代码 3-4：天天向上的力量（4）



实例代码 3.4　　　　e3.4DayDayUp365.py

```
1  #e3.4DayDayUp365.py
2  dayup, dayfactor = 1.0, 0.01
3  for i in range(365):
4      if i % 7 in [6, 0]:
5          dayup = dayup * (1 + dayfactor)
6      else:
7          dayup = dayup * (1 - dayfactor)
8  print("向上 5 天向下 2 天的力量: {:.2f}.".format(dayup))
```

运行结果如下。每周努力 5 天，而不是每天，一年下来，水平仅是初始的 4.63 倍！与每天坚持所提高的 37 倍相去甚远。

```
>>>
向上 5 天向下 2 天的力量: 4.63
```

【实例代码 3.5】天天向上。

如果对实例代码 3.4 的结果感到意外，那自然会有如下疑问：每周工作 5 天，休息 2 天，休息日水平下降 0.01，工作日要努力到什么程度，一年后的水平才与每天努力 1%取得的效果一样呢？

这个计算问题有很多种求解方法，这里采用通过多次运算求解问题的解决方案，即由程序从低到高逐渐增加每天努力的程度值，当运算结果刚刚超过 1.0 时，这个努力值就是上述问题的解。这种方案中，类似实例代码 3.4 的程序会被频繁调用，因此，可以通过保留字 def 定义一个求解天天向上的函数，这样将能更好地提升程序可读性。代码如下：

源代码 3-5：天天向上的力量（5）



实例代码 3.5　　　　e3.5DayDayUp365.py

```
1  #e3.5DayDayUp365.py
2  def dayUP(df):
3      dayup = 0.01
4      for i in range(365):
5          if i % 7 in [6, 0]:
6              dayup = dayup * (1 - 0.01)
7          else:
8              dayup = dayup * (1 + df)
9      return dayup
10 dayfacotr = 0.01
11 while (dayUP(dayfactor)<37.78):
```

```
12        dayfactor += 0.001
13    print("每天的努力参数是: {:.3f}.".format(dayfactor))
```

实例代码 3.5 第 9 行采用保留字 return 将函数 dayUP()的运行结果返回。程序运行结果如下：

```
>>>
每天的努力参数是: 0.019.
```

对比实例代码 3.3 和 3.5 的运行结果，每天努力 1%，坚持 365 天不间断，一年下来 1.0 的初始水平可以提高 37 倍！而如果每周连续努力 5 天，休息 2 天，为了达到每天努力 1%所达到的水平，则需要在工作日将提高的程度达到约 2%，即要努力 1 倍才仅是为了休息 2 天。这就是天天向上的力量！

拓展： GRIT：成功的关键

Grit 原义是砂砾，即砂堆中坚硬耐磨的颗粒。这里，GRIT 是 gritty（坚忍不拔的）的名词，由美国宾夕法尼亚大学心理学教授 Angela Duckworth 提出，与中文中“坚毅”的含义最为接近。Duckworth 教授曾经在纽约一所公立中学讲授初一数学，她很快发现智商并不是区分成功和那些挣扎但最后失败的学生的唯一标准。受这段经历启发，她于 2005 年在宾夕法尼亚大学开拓并发展了这个概念，奠定了 GRIT 研究的基础，研究表明：成功的先兆不是智商，而是日复一日的坚持，这是“坚毅”的力量。

GRIT 是对长期目标的持续激情及持久耐力，是不忘初衷、专注投入、坚持不懈，是一种包涵了自我激励、自我约束和自我调整的性格特征。GRIT 研究的重要价值在于它给教育发出警示：决定成功最重要的因素，不在于教育过程灌输了多少知识，而在于教育是否帮助学习者建立了“坚毅”的性格，后者则是成功的决定因素。Duckworth 教授的著作 *GRIT: The Power of Passion and Perseverance* 已经成为解释“坚毅”力量的经典作品，是心理学读物中最畅销的书籍。

早在 100 年前，被毛泽东主席誉为“华侨旗帜，民族光辉”的著名爱国华侨领袖、企业家、慈善家、曾经的华人首富陈嘉庚先生提出了“诚毅”二字，他在烽火战争年代坚持投身教育事业，先后创办了厦门大学、集美大学，用一生的实践诠释了获得成功的关键。“诚毅”，即“诚以待人，毅以处事”，被誉为“陈嘉庚精神”的核心，也成为集美大学的校训。

思考与练习

3.12　一年 365 天，初始水平值为 1.0，每工作一天水平增加 N，不工作时水平不下降，一周连续工作 4 天，请编写程序运算结果并填写下表：

N	0.001	0.002	0.003	0.004	0.005	0.006	0.007	0.008	0.009	0.010
年终值										

3.13 一年 365 天，初始水平值为 1.0，每工作一天水平增加 N，不工作时水平不下降，一周连续工作 5 天，请编写程序运算结果并填写下表。

N	0.001	0.002	0.003	0.004	0.005	0.006	0.007	0.008	0.009	0.010
年终值										

3.14 一年 365 天，初始水平值为 1.0，每工作一天水平增加 N，不工作时水平不下降，一周连续工作 6 天，请运算结果并填写下表。

N	0.001	0.002	0.003	0.004	0.005	0.060	0.007	0.008	0.009	0.010
年终值										

3.15 一年 360 天，初始水平值为 1.0，以每个月 30 天计算，在每个月月初连续 10 天里，每工作一天水平增加 N，该月其他时间工作与否都不增加水平值。请运算结果并填写下表。

N	0.001	0.002	0.003	0.004	0.005	0.006	0.007	0.008	0.009	0.010
年终值										

3.5 字符串类型及其操作

要点： 字符串是字符的序列表示，可以通过基本的字符串操作符、内置字符串处理函数和字符串处理方法等对字符串进行操作。

3.5.1 字符串类型的表示

字符串是字符的序列表示，可以由一对单引号（'）、双引号（"）或三引号（'''）构成。其中，单引号和双引号都可以表示单行字符串，两者作用相同。使用单引号时，双引号可以作为字符串的一部分；使用双引号时，单引号可以作为字符串的一部分。三引号可以表示单行或者多行字符串。3 种表示方式如下。

单引号字符串：'单引号表示，可以使用"双引号"作为字符串的一部分'
双引号字符串："双引号表示，可以使用'单引号'作为字符串的一部分"
三引号字符串：'''三引号表示可以使用"双引号"
'单引号'
也可以换行
'''

打印字符串的 Python 程序运行结果如下，注意其中的引号部分：

```
>>> print('单引号表示可以使用"双引号"作为字符串的一部分')
```

```
单引号表示可以使用"双引号"作为字符串的一部分
>>>print("双引号表示可以使用'单引号'作为字符串的一部分")
双引号表示可以使用'单引号'作为字符串的一部分
>>>print('''三引号中可以使用"双引号"
'单引号'
也可以使用换行''')
三引号中可以使用"双引号"
'单引号'
也可以使用换行
```

input()函数将用户输入的内容当作一个字符串类型，这是获得用户输入的常用方式。print()函数可以直接打印字符串，这是输出字符串的常用方式。如下例子展示了如何用变量 name 来存储用户的名字，再输出这个变量的内容：

```
>>>name = input("请输入名字: ")
请输入名字: Python 语言
>>>print(name)
Python 语言
```

2.2.4 节已经介绍，字符串包括两种序号体系：正向递增序号和反向递减序号。如果字符串长度为 L，正向递增需要以最左侧字符序号为 0，向右依次递增，最右侧字符序号为 L-1；反向递减序号以最右侧字符序号为–1，向左依次递减，最左侧字符序号为–L。这两种索引字符的方法可以在一个表示中使用。

Python 字符串也提供区间访问方式，采用[N: M]格式，表示字符串中从 N 到 M（不包含 M）的子字符串，其中，N 和 M 为字符串的索引序号，可以混合使用正向递增序号和反向递减序号。如果表示中 M 或者 N 索引缺失，则表示字符串把开始或结束索引值设为默认值。

字符串以 Unicode 编码存储，因此，字符串的英文字符和中文字符都算作 1 个字符。观察下面实例：

```
>>>name = "Python 语言程序设计"
>>>name[0]
'P'
>>>print(name[0], name[7], name[-1])
P 言 计
>>>print(name[2:-4])
thon 语言
>>>print(name[:6])
Python
>>>print(name[6:])
语言程序设计
>>>print(name[:])
Python 语言程序设计
```

反斜杠字符（\）是一个特殊字符，在字符串中表示转义，即该字符与后面相邻

的一个字符共同组成了新的含义。例如，\n 表示换行、\\表示反斜杠、\'表示单引号、\" 表示双引号、\t 表示制表符（Tab）等。例如：

```
>>>print("Python\n 语言\t 程序\t 设计")
Python
语言	程序	设计
```

3.5.2 基本的字符串操作符

Python 提供了 5 个字符串的基本操作符，如表 3.10 所示。

表 3.10 基本的字符串操作符（共 5 个）

操 作 符	描 述
x + y	连接两个字符串 x 与 y
x * n 或 n * x	复制 n 次字符串 x
x in s	如果 x 是 s 的子串，返回 True，否则返回 False
str[i]	索引，返回第 i 个字符
str[*N*: *M*]	切片，返回索引第 *N* 到第 *M* 的子串，其中不包含 *M*

与字符串操作符有关的实例如下：

```
>>>"Python 语言" + "程序设计"
'Python 语言程序设计'
>>>name = "Python 语言" + "程序设计" + "基础"
>>>name
'Python 语言程序设计基础'
>>>"GOAL!" * 3
'GOAL!GOAL!GOAL!'
>>>"Python 语言" in name
True
>>>'Y' in "Python 语言"
False
```

源代码 3-6：获取星期字符串



【微实例 3.1】获取星期字符串。

程序读入一个表示星期几的数字（1～7），输出对应的星期字符串名称。例如，输入 3，返回“星期三”。代码如下：

微实例 3.1　　m3.1PrintWeekname.py

```
1  weekstr = "星期一星期二星期三星期四星期五星期六星期日"
2  weekid = eval(input("请输入星期数字(1-7): "))
3  pos = (weekid - 1)*3
4  print(weekstr[pos: pos+3])
```

程序运行结果如下：

```
>>>
请输入星期数字(1-7)：3
星期三
```

微实例 3.1 通过在字符串中截取适当子串实现星期名称的查找。问题的关键在于找出子串的剪切位置。因为每个星期日期的缩写都由 3 个字符组成，如果知道星期日期字符串的起始位置，就能很容易获得缩写子串。通过下面语句，可以获得从起始位置 pos 开始且长度为 3 的子串：

```
weekAbbr = weekstr[pos: pos+3]
```

使用字符串作为查找表的缺点是，所剪切的子字符串长度必须相同。如果各缩写表示长度不同，还需要其他语句辅助。例如，请读者思考，该如何实现一个“获取月份字符串”呢？要求根据 1～12 的数字返回月份名称。

拓展： 特殊的格式化控制字符

字符串中可以增加特殊的格式化控制字符，用来输出特殊效果。特殊的格式化控制字符使用反斜杠（\）开头，常用控制字符如下。

\a：蜂鸣，响铃。

\b：回退，向后退一格。

\f：换页。

\n：换行，光标移动到下行首行。

\r：回车，光标移动到本行首行。

\t：水平制表。

\v：垂直制表。

\0：NULL，什么都不做。

将这些控制字符放到字符串中，显示效果如下：

```
>>> print("爱\t生活\t爱\tPython")
爱      生活      爱      Python
```

需要注意的是，IDLE 开发环境不支持部分特殊控制字符，比如\b 和\r 等，使用这些控制符的程序需要编写代码保存为 py 文件，然后在命令行下执行。

3.5.3　内置的字符串处理函数

Python 解释器提供了一些内置函数，详细请参考 5.8 节。其中，有 6 个函数与字符串处理相关，如表 3.11 所示。

len(x)返回字符串 x 的长度，Python 3 以 Unicode 字符为计数基础，因此，字符串中英文字符和中文字符都是 1 个长度单位。

```
>>>len("Python 语言程序设计")
12
```

表 3.11 内置的字符串处理函数（共 6 个）

函 数	描 述
len(x)	返回字符串 x 的长度，也可返回其他组合数据类型元素个数
str(x)	返回任意类型 x 所对应的字符串形式
chr(x)	返回 Unicode 编码 x 对应的单字符
ord(x)	返回单字符表示的 Unicode 编码
hex(x)	返回整数 x 对应十六进制数的小写形式字符串
oct(x)	返回整数 x 对应八进制数的小写形式字符串

str(x)返回 x 的字符串形式，其中，x 可以是数字类型或其他类型，例如：

```
>>>str(3.1415926)
'3.1415926'
```

每个字符在计算机中可以表示为一个数字，称为编码。字符串则以编码序列方式存储在计算机中。目前，计算机系统使用的一个重要编码是 ASCII 编码，该编码用数字 0～127 表示计算机键盘上的常见字符以及一些被称为控制代码的特殊值。例如，大写字母 A～Z 用 65～90 表示，小写字母 a～z 用 97～122 表示。

ASCII 编码针对英语字符设计，它没有覆盖其他语言存在的更广泛字符，因此，现代计算机系统正逐步支持一个更大的编码标准 Unicode，它支持几乎所有书写语言的字符。Python 字符串中每个字符都使用 Unicode 编码表示。

chr(x)和 ord(x)函数用于在单字符和 Unicode 编码值之间进行转换。chr(x)函数返回 Unicode 编码对应的字符，其中，Unicode 编码 x 的取值范围是 0 到 1 114 111（即十六进制数 0x10FFFF）。ord(x)函数返回单字符 x 对应的 Unicode 编码。例如：

```
>>>"1 + 1 = 2 " + chr(10004)
'1 + 1 = 2 ✔'
>>>"金牛座字符♉的 Unicode 值是：" + str(ord("♉"))
'金牛座字符♉的 Unicode 值是：9801'
```

hex(x)和 oct(x)函数分别返回整数 x 对应十六进制和八进制值的字符串形式，字符串以小写形式表示。例如：

```
>>>hex(255)
'0xff'
>>>oct(-255)
'-0o377'
```

【微实例 3.2】恺撒密码。

设想在某些情况下给朋友传递字条信息，但又不希望传递中途被第三方看懂这些信息，因此需要对字条信息进行加密处理。传统加密算法很多，这里介绍一种非常简单的加密算法——凯撒密码。凯撒密码是古罗马凯撒大帝用来对军事情报进行加密的算法，它采用了替换方法对信息中的每一个英文字符循环替换为字母表序列中该字符后面第三个字符，对应关系如下。

原文：A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

密文：D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

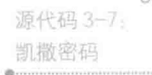
原文字符 P，其密文字符 C 满足如下条件：

$C = (P + 3) \bmod 26$

解密方法反之，满足：

$P = (C - 3) \bmod 26$

假设用户可能使用的信息仅包括小写字母 a～z，则该微实例对应的加密代码如下：





微实例 3.2　　m3.2 CaesarCode.py

```
1  plaincode = input("请输入明文： ")
2  for p in plaincode:
3      if ord("a") <= ord(p) <= ord("z"):
4          print(chr(ord("a") + (ord(p) - ord("a") + 3)%26), end='')
5      else:
6          print(p, end='')
```

微实例运行结果如下：

```
>>>
请输入明文: python is an excellent language.
sbwkrq lv dq hafhoohqw odqjxdjh.
```

请读者参考微实例 3.2 编写凯撒密码的解密程序。另外，6.7 节将介绍一个与恺撒密码相似的加解密算法，代码更为精炼。

3.5.4　内置的字符串处理方法

在 Python 解释器内部，所有数据类型都采用面向对象方式实现，封装为一个类。字符串也是一个类，它具有类似<a>.<b>()形式的字符串处理函数。在面向对象中，这类函数被称为“方法”。字符串类型共包含 43 个内置方法。鉴于部分内置方法并不常用，限于篇幅，这里仅介绍其中 16 个常用方法，如表 3.12 所示（其中 str 代表字符串或变量）。

表 3.12　常用的内置字符串处理方法（共 16 个）

方　法	描　述
str.lower()	返回字符串 str 的副本，全部字符小写
str.upper()	返回字符串 str 的副本，全部字符大写
str.islower()	当 str 所有字符都是小写时，返回 True，否则返回 False
str.isprintable()	当 str 所有字符都是可打印的，返回 True，否则返回 False
str. isnumeric()	当 str 所有字符都是数字时，返回 True，否则返回 False

续表

方 法	描 述
str.isspace()	当 str 所有字符都是空格，返回 True，否则返回 False
str.endswith(suffix[,start[,end]])	str[start: end] 以 suffix 结尾返回 True，否则返回 False
str.startswith(prefix[, start[, end]])	str[start: end] 以 prefix 开始返回 True，否则返回 False
str.split(sep=None, maxsplit=-1)	返回一个列表，由 str 根据 sep 被分隔的部分构成
str.count(sub[,start[,end]])	返回 str[start: end]中 sub 子串出现的次数
str.replace(old, new[, count])	返回字符串 str 的副本，所有 old 子串被替换为 new，如果 count 给出，则前 count 次 old 出现被替换
str.center(width[, fillchar])	字符串居中函数，详见函数定义
str.strip([chars])	返回字符串 str 的副本，在其左侧和右侧去掉 chars 中列出的字符
str.zfill(width)	返回字符串 str 的副本，长度为 width，不足部分在左侧添 0
str.format()	返回字符串 str 的一种排版格式，3.6 节将详细介绍
str.join(iterable)	返回一个新字符串，由组合数据类型（见第 6 章）iterable 变量的每个元素组成，元素间用 str 分隔

str.split(sep=None, maxsplit=-1)方法返回一个列表，列表是一种存储多个数据的数据类型，6.2 节将详细介绍，其中，分隔 str 的标识符是 sep，默认分隔符为空格。如果给出 maxsplit 参数，则只分隔前 maxsplit 个字符，默认 maxsplit 参数可以不给出。

str.center(width[, fillchar])方法返回长度为 width 的字符串，其中，str 处于新字符串中心位置，两侧新增字符采用 fillchar 填充，当 width 小于字符串长度时，返回 str。

str.zfill(width)方法返回长度为 width 的字符串，如果字符串长度不足 width，在左侧添加字符"0"，但如果 str 最左侧是字符"+"或者"-"，则从第二个字符左侧添加"0"，当 width 小于字符串长度时，返回 str。该方法主要用于格式化数字形字符串中。

表 3.12 中给出的方法在字符串处理时十分有用，这里仅给出少数例子，请读者在应用中逐步实践。

```
>>>hex(255)
'0xff'
>>>oct(-255)
'-0o377'
>>>"Python is an excellent language.".split()
['Python','is','an','excellent','language.]
>>>"PYTHON".center(40,'=')
'=================PYTHON================='
>>>"123".zfill(40)
'0000000000000000000000000000000000000123'
>>>"-123".zfill(40)
'-000000000000000000000000000000000000123'
```

思考与练习

3.16 s="hello"，t="world"，s+=t，则 s、s[-1]、s[2:8]、s[::3]、s[-2::-1]分别是

多少？

3.17　判断题：Python 中"4"+"5"结果为"9"。

3.18　采用微实例 3.1 的设计思路还能完成哪些常用计算需求？

3.19　s="Python String"，写出下列操作的输出结果：

```
s.upper()、s.lower()、s.find('i')
s.replace('ing','gni')、s.split(' ')
```

3.20　下列表达式错误的是（　）。

A．'abcd'<'ad'　　B．'abc'<'abcd'

C．''<'a'　　D．'Hello'>'hello'

3.6　字符串类型的格式化

要点：字符串通过 format()方法进行格式化处理。

为什么会有字符串类型的格式化问题呢？例如，一个程序希望输出如下内容：

“2016-12-31: 计算机 PYTHON 的 CPU 占用率为 10%。”

其中，下画线内容可能会变化，需要由特定函数运算结果进行填充，最终形成上述格式字符串作为输出结果。字符串格式化用于解决字符串和变量同时输出时的格式安排。

字符串是程序向控制台、网络、文件等介质输出运算结果的主要形式之一，为了能提供更好的可读性和灵活性，字符串类型的格式化是运用字符串类型的重要内容之一。Python 语言同时支持两种字符串格式化方法，一种类似 C 语言中 printf()[1] 函数的格式化方法，支持该方法主要考虑与大批 C 语言程序员编程习惯相一致；另一种采用专门的 str.format()格式化方法。由于 Python 中更为接近自然语言的复杂数据类型（如列表和字典等）无法通过类 C 的格式化方法很好表达，Python 已经不在后续版本中改进 C 风格格式化方法。Python 语言将主要采用 format()方法进行字符串格式化。本书所有例子均采用该方法，建议读者尽量使用该方法。

3.6.1　format()方法的基本使用

字符串 format()方法的基本使用格式如下：

```
<模板字符串>.format(<逗号分隔的参数>)
```

模板字符串由一系列槽组成，用来控制修改字符串中嵌入值出现的位置，其基本思想是将 format()方法中逗号分隔的参数按照序号关系替换到模板字符串的槽中。槽用大括号（{}）表示，如果大括号中没有序号，则按照出现顺序替换，如图 3.2

[1] printf()函数是 C 语言中向控制台输出信息的主要函数，类似 Python 语言中的 print()函数。

所示。如果大括号中指定了使用参数的序号，按照序号对应参数替换，如图 3.3 所示，参数从 0 开始编号。调用 format()方法后会返回一个新的字符串。例如：

```
>>>"{}：计算机{}的 CPU 占用率为{}%。".format("2016-12-31","PYTHON",10)
'2016-12-31：计算机 PYTHON 的 CPU 占用率为 10%。'
```

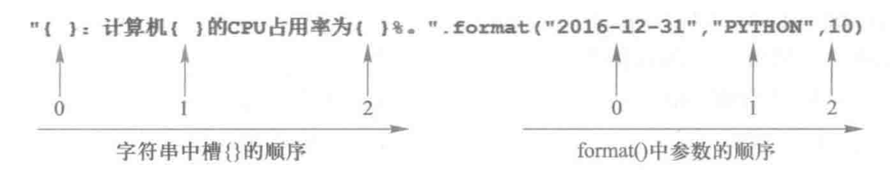


图 3.2 format()方法的槽顺序和参数顺序

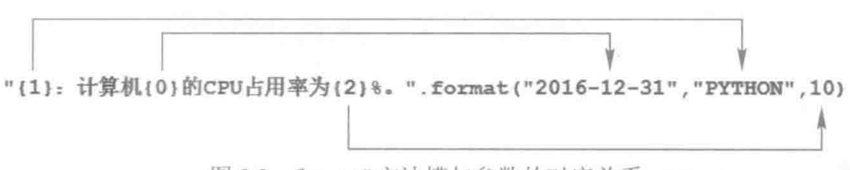


图 3.3 format()方法槽与参数的对应关系

format()方法可以非常方便地连接不同类型的变量或内容，如果需要输出大括号，采用{{表示{，}}表示}，例如：

```
>>>"{}{}{}".format("圆周率是",3.1415926,"...")
'圆周率是 3.1415926...'
>>>"圆周率{{{1}{2}}}是{0}".format("无理数",3.1415926,"...")
'圆周率{3.1415926...}是无理数'
>>>s="圆周率{{{1}{2}}}是{0}"                  #大括号本身是字符串的一部分
>>>s
'圆周率{{{1}{2}}}是{0}'
>>>s.format("无理数",3.1415926,"...")        #当调用 format()时解析大括号
'圆周率{3.1415926...}是无理数'
```

3.6.2 format()方法的格式控制

format()方法中模板字符串的槽除了包括参数序号，还可以包括格式控制信息。此时，槽的内部样式如下：

```
{<参数序号>: <格式控制标记>}
```

其中，格式控制标记用来控制参数显示时的格式，格式内容如图 3.4 所示。

:	<填充>	<对齐>	<宽度>	<,>	<.精度>	<类型>
引导符号	用于填充的单个字符	< 左对齐 > 右对齐 ^ 居中对齐	槽的设定输出宽度	数字的千位分隔符 适用于整数和浮点数	浮点数小数部分的精度 或 字符串的最大输出长度	整数类型 b,c,d,o,x,X 浮点数类型 e,E,f,%

图 3.4 槽中格式控制标记的字段

格式控制标记包括<填充>、<对齐>、<宽度>、<，>、<.精度>、<类型>6 个字段，这些字段都是可选的，可以组合使用，这里按照使用方式逐一介绍。

<宽度>、<对齐>和<填充>是 3 个相关字段。<宽度>指当前槽的设定输出字符宽度，如果该槽对应的 format()参数长度比<宽度>设定值大，则使用参数实际长度；如果该值的实际位数小于指定宽度，则位数将被默认以空格字符补充。<对齐>指参数在宽度内输出时的对齐方式，分别使用<、>和^ 3 个符号表示左对齐、右对齐和居中对齐。<填充>指宽度内除了参数外的字符采用什么方式表示，默认采用空格，可以通过填充更换。例如：

```
>>>s = "PYTHON"
>>>"{0:30}".format(s)                    #默认左对齐
'PYTHON                        '
>>>"{0:>30}".format(s)                   #右对齐
'                        PYTHON'
>>>"{0:*^30}".format(s)                  #居中且使用*填充
'************PYTHON************'
>>>"{0:-^30}".format(s)                  #居中且使用-填充
'------------PYTHON------------'
>>>"{0:3}".format(s)
'PYTHON'
```

格式控制标记中的逗号（,）用于显示数字类型的千位分隔符，例如：

```
>>>"{0:-^20,}".format(1234567890)
'---1,234,567,890----'
>>>"{0:-^20}".format(1234567890)    #对比输出
'-----1234567890-----'
>>>"{0:-^20,}".format(12345.67890)
'----12,345.6789-----'
```

<.精度>表示两个含义，由小数点（.）开头。对于浮点数，精度表示小数部分输出的有效位数。对于字符串，精度表示输出的最大长度。

```
>>>"{0:.2f}".format(12345.67890)
'12345.68'
>>>"{0:H^20.3f}".format(12345.67890)
'HHHHH12345.679HHHHHH'
>>>"{0:.4}".format("PYTHON")
'PYTH'
```

<类型>表示输出整数和浮点数类型的格式规则。对于整数类型，输出格式包括以下 6 种。

（1）b：输出整数的二进制方式。

（2）c：输出整数对应的 Unicode 字符。

（3）d：输出整数的十进制方式。

（4）o：输出整数的八进制方式。

（5）x：输出整数的小写十六进制方式。

（6）X：输出整数的大写十六进制方式。

```
>>>"{0:b},{0:c},{0:d},{0:o},{0:x},{0:X}".format(425)
'110101001,Σ,425,651,1a9,1A9'
```

对于浮点数类型，输出格式包括以下 4 种。

（1）e：输出浮点数对应的小写字母 e 的指数形式。

（2）E：输出浮点数对应的大写字母 E 的指数形式。

（3）f：输出浮点数的标准浮点形式。

（4）%：输出浮点数的百分形式。

浮点数输出时尽量使用<.精度>表示小数部分的宽度，有助于更好控制输出格式。

```
>>>"{0:e},{0:E},{0:f},{0:%}".format(3.14)
'3.140000e+00,3.140000E+00,3.140000,314.000000%'
>>>"{0:.2e},{0:.2E},{0:.2f},{0:.2%}".format(3.14)
'3.14e+00,3.14E+00,3.14,314.00%'
```

拓展： 字符串和字节流

字节流是字节组成的序列，字节由固定的 8 个比特组成，因此，字节流从二进制角度有确定的长度和存储空间。Python 字符串由编码字符的序列组成，字符根据编码不同长度也不相同。因此，从存储空间角度，字符串和字节流不相同。

硬盘上所有文件都以字节形式存储，例如，文本、图片及视频等，真正存储和传输数据时都是以字节为单位。字符值在内存中形成，由字节流经过编码处理后产生。

阶段测试 3-3：Python 字符串类型小测验



思考与练习

3.21 请思考并描述下面 Python 语句的输出结果：

```
print("{:>15s}:{:<8.2f}".format("Length",23.87501))
```

3.22 格式化输出 389 的二进制、八进制、十进制、十六进制的表达形式，以及对应的 Unicode 字符。

3.23 格式化输出 0.002 178 对应的科学表示法形式，保留 4 位有效位的标准浮点形式以及百分形式。

程序练习 3-3：半小时学 Python 字符串类型



3.7 实例 4：文本进度条

要点： 这是一个利用格式化输出和时间延迟实现控制台风格文本进度条的实例。

3.7.1　简单的开始

进度条是计算机处理任务或执行软件中常用的增强用户体验的重要手段，它能够实时显示任务或软件的执行进度。本节将利用 Python 字符串处理方法实现文本进度条功能。

最简单地，利用 print()函数实现简单的非刷新文本进度条。基本思想是按照任务执行百分比将整个任务划分为 100 个单位，每执行 *N*%输出一次进度条。每一行输出包含进度百分比，代表已完成的部分（**）和未完成的部分（..）的两种字符，以及一个跟随完成度前进的小箭头，风格如下：

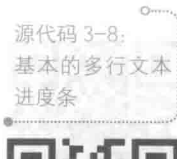
```
%10 [*****->.......................................]
```

由于程序执行速度远超过人眼的视觉停留时间，直接进行字符输出几乎是瞬间完成，不利于观察。为了模拟任务处理的时间效果，调用 Python 标准时间库 time，使用 time.sleep(t)函数将当前程序暂时挂起 t s，t 可以是小数。由此可以接近真实的模拟进度条效果输出。使用 import 保留字调用 time 库。

```
>>> import time
```

默认情况，print()函数在输出结尾处会自动产生一个'\n'，即换行符，从而让光标自动移动到下一行行首，这样上一步输出依旧保存在界面上。

采用 for 循环和 print()函数构成程序的主体部分，输出百分比最高（100%）为 3 位数据，为了使输出显得整齐，可以使用{:^3.0f}格式化百分比部分。这个简单的文本进度条代码如下。变量 scale 表示输出进度条的精度，读者可以修改这个值观察效果变化。





实例代码 4.1　　e4.1TextProgressBar.py

```
1   #e4.1TextProgress Bar.py
2   import time
3   scale = 10
4   print("------执行开始------")
5   for i in range(scale+1):
6       a, b = '**' * i,'..' * (scale - i)
7       c = (i/scale)*100
8       print("%{:^3.0f}[{}->{}]" .format (c, a, b))
9       time.sleep(0.1)
10  print("------执行结束------")
```

程序的输出效果如下：

```
------执行开始------
```

```
% 0 [->...................]
%10  [**->..................]
%20  [****->................]
%30   [******->..............]
%40   [********->............]
%50    [**********->..........]
%60    [************->........]
%70     [**************->......]
%80     [****************->....]
%90      [******************->..]
%100     [********************->]
------执行结束------
```

3.7.2 单行动态刷新

常用的计算机系统中都有进度条，这些进度条一般只在一行中改变进度比例，与实例代码 4.1 相比，区别在于原地输出和动态刷新，其基本思想是将每一次进度输出都固定在同一行，并不断地用新生成的字符串覆盖之前的输出，形成进度条不断刷新的动态效果。这种效果称为“单行动态刷新”，可以通过 print()函数实现。

采用 print()函数的具体方法是，在 print()函数中更换参数 end 的默认值为'',即每次使用 print()函数输出时不换行。此时，系统输出指针还停留在上一次输出的行尾，下一次输出在字符串前部增加转义符'\r'，该转义符把输出指针移动到行首而不换行。动态刷新一个百分比的完整代码如下：

源代码 3-9：单行动态刷新

实例代码 4.2　　e4.2TextProgressBar.py

```
1  #e4.2TextProgressBar.py
2  import time
3  for i in range(101):
4      print("\r{:2}%".format(i), end="")
5      time.sleep(0.05)
```

上述程序在 IDLE 中的执行效果如下。

```
>>>
 0%  1%  2%  3%  4%  5%  6%  7%  8%  9% 10% 11% 12% 13% 14% 15% 16% 17%
18% 19% 20% 21% 22% 23% 24% 25% 26% 27% 28% 29% 30% 31% 32% 33% 34% 35%
36% 37% 38% 39% 40% 41% 42% 43% 44% 45% 46% 47% 48% 49% 50% 51% 52% 53%
54% 55% 56% 57% 58% 59% 60% 61% 62% 63% 64% 65% 66% 67% 68% 69% 70% 71%
72% 73% 74% 75% 76% 77% 78% 79% 80% 81% 82% 83% 84% 85% 86% 87% 88% 89%
90% 91% 92% 93% 94% 95% 96% 97% 98% 99%100%
```

为什么输出没有单行刷新呢？这是因为 IDLE 本身屏蔽了单行刷新功能，如果希望获得刷新效果，请使用控制台的命令行执行 e4.2TextProgressBar.py 程序。以 Windows 系统为例，启动命令行工具（<Windows 系统安装目录>\system32\cmd.exe），选择到 e4.2TextProgressBar.py 文件所在目录执行如下命令行，执行完成后即可输出可以单行刷新且快速变化的百分比。

```
:\>python e4.2TextProgressBar.py
100%
```

3.7.3　带刷新的文本进度条

将前两小节的程序合并，再添加开始和结束提示语，可以很好地实现带刷新的文本进度条。为了进一步提高用户体验，在文本进度条中增加运行时间的监控，这里采用 time 库中的 time.clock()函数。time.clock()函数一般多次出现，第一次调用时计时开始，同一程序中第二次及后续调用时返回与第一次计时之间的时间差，单位为秒。该函数主要用来统计程序运行时间，增加用户体验。文本进度条完整代码如下：

实例代码 4.3　　　　　e4.3TextProgressBar.py

源代码 3-10：带刷新的文本进度条



```
1   #e4.3TextProgress Bar.py
2   import time
3   scale = 50
4   print("执行开始".center(scale//2,'-'))
5   t = time.clock()
6   for i in range(scale+1):
7       a = '*' * i
8       b = '.' * (scale - i)
9       c = (i/scale)*100
10      t -=  time.clock()
11      print("\r{:^3.0f}%[{}->{}]{:.2f}s".format(c,a,b,-t),\
            end='')
12      time.sleep(0.05)
13  print("\n"+"执行结束".center(scale//2,'-'))
```

采用命令行执行上述文件，效果如下：

```
:\>python e4.3TextProgressBar.py
-----------执行开始----------
100%[**************************************************->] 65.71s
-----------执行结束----------
```

拓展： 进度条设计方法

通常情况下，进度条会通过线性增长来表示一项工作的完成进度。然而，现实中可能出现各种原因导致进度条不能保持线性增长。不同进度条会影响人们对时间的感知。美国卡耐基·梅隆大学的研究人员做了进度条设计和人类心理的相关研究[4]。

该研究首先列举了 9 个不同的进度条设计函数，如表 3.13 所示，并根据这些函数的实际执行进度和进度条的显示进度进行了比较，如图 3.5 所示，可以看出，这些函数显示实际进度的方式五花八门，各有各的特色。

表 3.13 进度条设计函数

设计名称	趋势	设计函数
Liner	Constant	$f(x) = x$
Early Pause	Speeds up	$f(x) = x+(1-\sin(x*\pi*2+\pi/2))/-8$
Late Pause	Slows down	$f(x) = x+(1-\sin(x*\pi*2+\pi/2))/8$
Slow Wavy	Constant	$f(x) = x+\sin(x*\pi*5)/20$
Fast Wavy	Constant	$f(x) = x+\sin(x*\pi*20)/80$
Power	Speeds up	$f(x) = (x+(1-x)*0.03)^2$
Inverse Power	Slows down	$f(x) = 1+(1-x)^{1.5}*-1$
Fast Power	Speeds up	$f(x) = (x+(1-x)/2)^8$
Inverse Fast Power	Slows down	$f(x) = 1+(1-x)^3*-1$

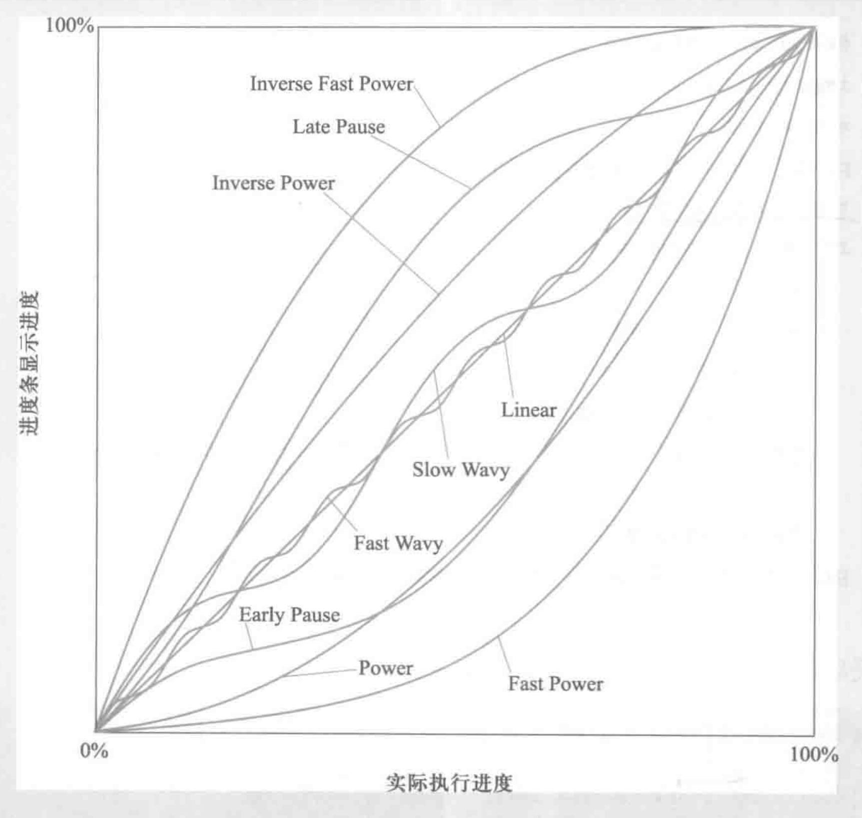


图 3.5 9 个进度条设计函数的显示曲线

研究人员进一步根据这些函数进行了心理学实验，结果表明：

（1）用户可以容忍最初的慢速增长。

（2）用户难以容忍在进程快要结束时进度条突然停滞不前。

上述研究实际上建议在设计进度条时，可以适当降低开始阶段的进展速度，然后适当加快末尾阶段的进展速度，这样的调整可以给用户带来更舒适、“更快”的体验。

思考与练习

3.24 进度条反映了软件的执行速度，请思考并给出至少 3 种提高软件执行速度的方法。

3.25 str.center()方法的功能是什么？

3.26 如果将\r 放在 print()中字符串的其他部分，会产生什么效果？

本章小结

本章首先介绍了计算机中常用的数字类型及操作，包括 Python 内置的数值运算操作和数字类型转换函数等，进一步介绍了常用的数学计算标准库 math 库。采用数学计算将模糊的“好好学习，天天向上”数据化，展示了持续性学习的强大力量。本章同时介绍了字符串类型及其操作和格式化方法，并通过字符串格式化实现控制台风格的文本进度条。

程序练习题

程序练习 3-4：章节程序练习题



3.1 重量计算。月球上物体的体重是在地球上的 16.5%，假如你在地球上每年增长 0.5 kg，编写程序输出未来 10 年你在地球和月球上的体重状况。

3.2 天天向上续。尽管每天坚持，但人的能力发展并不是无限的，它符合特定模型。假设能力增长符合如下带有平台期的模型：以 7 天为周期，连续学习 3 天能力值不变，从第 4 天开始至第 7 天每天能力增长为前一天的 1%。如果 7 天中有 1 天间断学习，则周期从头计算。请编写程序回答，如果初识能力值为 1，连续学习 365 天后能力值是多少？

3.3 天天向上续。采用程序练习题 3.2 的能力增长模型，如果初始能力值为 1，固定每 10 天休息 1 天，365 天后能力值是多少？如果每 15 天休息 1 天呢？

3.4 回文数判断。设 n 是一任意自然数，如果 n 的各位数字反向排列所得自然数与 n 相等，则 n 被称为回文数。从键盘输入一个 5 位数字，请编写程序判断这个

数字是不是回文数。

3.5 田字格的输出。使用 print()函数输出如图 3.6 所示样式的田字格。

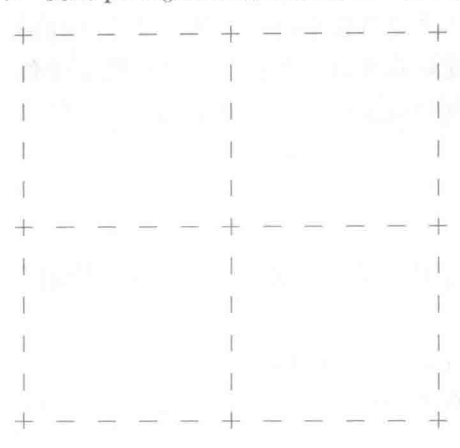


图 3.6 田字格效果

3.6 文本进度条。仿照实例 4，打印如下形式的进度条。

```
Starting … Done!
```

3.7 文本风格。将如下程序段存成文件，在控制台终端（如 Windows 的 cmd.exe）。运行该程序，观察输出效果。更改 print()函数的参数，例如，去掉 end 的赋值，再观察运行结果。

```
1  while True:
2      for i in ["/" , "-", "|", "\\", "|"]:
3          print("%s\r" % i ,end = '')
```

3.8 小巧而精致的第三方进度条工具库。tqdm 是一个快速、扩展性强的进度条工具库。tqdm 是一个第三方库，首先需要安装，然后才能使用。本书 8.6 节将详细介绍第三方库的安装方法。请读者提前参考阅读。运行如下程序，观察运行结果。

程序练习题代码 3.8　　a3.8tqdmBar.py

```
1  #a3.8tqdmBar.py
2  from tqdm import tqdm
3  from time import sleep
4  for i in tqdm(range(1,100)):
5      sleep(0.01)
```

第 4 章　程序的控制结构

电子教案 4-1：
程序的控制结构



好软件的作用是让复杂的东西看起来简单。

The function of good software is to make the complex appear to be simple.

——格雷迪·布奇(Grady Booch)

UML 和 Booch 方法的创始人

学习目标

（1）了解程序的基本结构并绘制流程图。
（2）掌握程序的分支结构。
（3）运用 if 语句实现分支结构。
（4）掌握程序的循环结构。
（5）运用 for 语句和 while 语句实现循环结构。
（6）掌握随机库的使用方法。
（7）了解程序的异常处理及用法。

对 π 的精确求解是数学历史上一直难以解决的问题之一，古希腊大数学家阿基米德使用迭代算法和两侧数值逼近的概念率先求出 π 小数点后 6 位精确值。公元 263 年，中国数学家刘徽用“割圆术”计算圆周率，“割之弥细，所失弥少，割之又割，以至于不可割，则与圆周合体而无所失矣。”，这其中蕴含了求极限的创新思想。2015 年，罗切斯特大学的科学家们在氢原子能级的量子力学计算中发现了圆周率相同的公式。

对于这个神秘的数字，用编程语言能获得什么新的认识呢？

4.1 程序的基本结构

> **要点：** 程序由 3 种基本结构组成：顺序结构、分支结构和循环结构。

4.1.1 程序流程图

程序流程图用一系列图形、流程线和文字说明描述程序的基本操作和控制流程，它是程序分析和过程描述的最基本方式。流程图的基本元素包括 7 种，如图 4.1 所示。

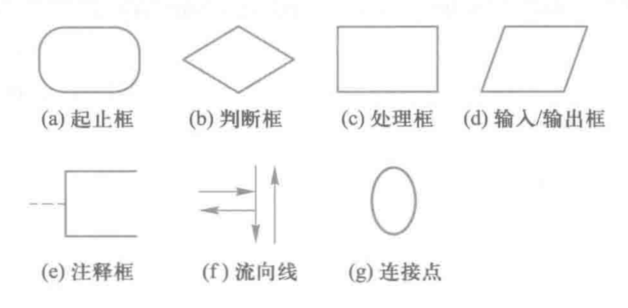


图 4.1 程序流程图的 7 种元素

其中，起止框表示一个程序的开始和结束；判断框判断一个条件是否成立，并根据判断结果选择不同的执行路径；处理框表示一组处理过程；输入/输出框表示数据输入或结果输出；注释框增加程序的解释；流向线以带箭头直线或曲线形式指示程序的执行路径；连接点将多个流程图连接到一起，常用于将一个较大流程图分隔为若干部分。图 4.2 所示为一个流程图示例，为了便于描述，采用连接点 A 将流程图分成两个部分。

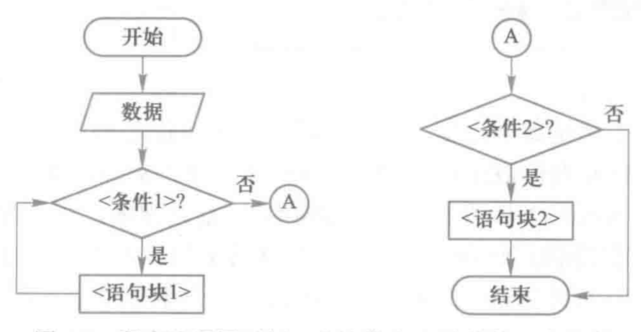


图 4.2 程序流程图示例：由连接点 A 连接的一个程序

4.1.2 程序的基本结构

目前为止，计算机程序可以看作是一条一条顺序执行的代码。顺序结构是程序

的基础，但单一的顺序结构不可能解决所有问题，因此需要引入控制结构来更改程序的执行顺序以满足多样的功能需求。

程序由 3 种基本结构组成：顺序结构、分支结构和循环结构。这些基本结构都有一个入口和一个出口。任何程序都由这 3 种基本结构组合而成。为了直观展示程序结构，这里采用流程图方式描述。

顺序结构是程序按照线性顺序依次执行的一种运行方式，如图 4.3 所示，其中语句块 1 和语句块 2 表示一个或一组顺序执行的语句。

分支结构是程序根据条件判断结果而选择不同向前执行路径的一种运行方式，如图 4.4 所示，根据分支路径上的完备性，分支结构包括单分支结构和二分支结构，二分支结构组合形成多分支结构。

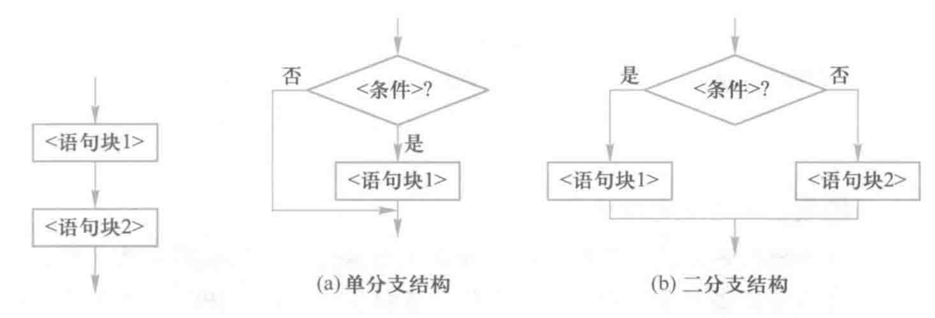


图 4.3　顺序结构的流程图表示　　　　图 4.4　分支结构的流程图表示

循环结构是程序根据条件判断结果向后反复执行的一种运行方式，如图 4.5 所示，根据循环体触发条件不同，循环结构包括条件循环和遍历循环结构。

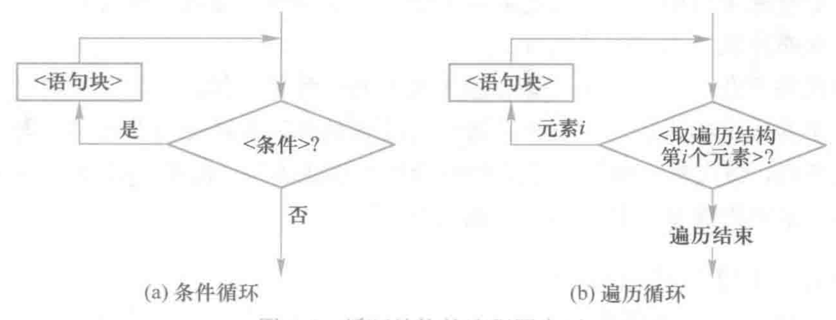


图 4.5　循环结构的流程图表示

4.1.3　程序的基本结构实例

对于一个计算问题，可以用 IPO、流程图或者直接以 Python 代码方式描述。本书仅对这几种描述进行介绍，功能简单的问题建议读者直接编写 Python 代码，功能复杂的问题可以采用 IPO 描述或流程图描述为手段。下面给出 3 个微实例，通过不同的描述方法具体解释程序的 3 种基本结构。

【微实例 4.1】圆面积和周长的计算。

根据圆的半径计算圆的面积和周长。图 4.6 分别给出了该问题的 IPO 描述、流

程图描述和 Python 代码描述。

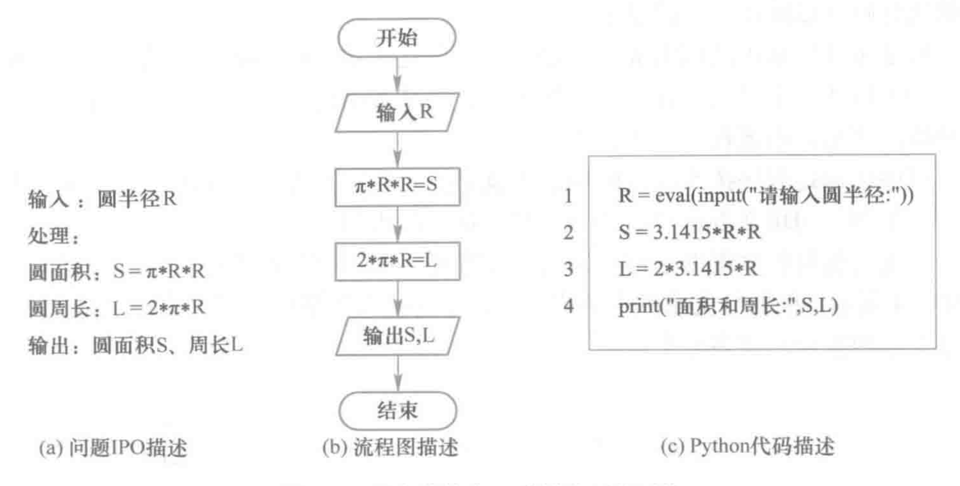


图 4.6 顺序结构在三种描述下的对比

拓展： 程序的描述方式

程序的描述方式主要有 3 类，分别是自然语言、流程图和伪代码。

自然语言描述方式指使用人类语言直接描述程序，IPO 描述是其中一种。优点是灵活自然，缺点是容易出现二义性，即一个描述可以产生多种不同的程序代码。

流程图描述是程序最直观易懂的表达方式，主要适用于较短的算法。优点是直观、清晰且逻辑确定，缺点是流程图绘制比较烦琐，当程序较大时流程图会很复杂，反而降低了表达的清晰性。

伪代码是介于自然语言与编程语言之间的一种算法描述语言。使用伪代码不用拘泥于具体编程语言，对整个算法运行过程的描述最接近自然语言。与自然语言描述不同，伪代码在保持程序结构的情况下描述算法。由于 Python 语言语法相对简单，本书没有采用伪代码方式描述程序。

【微实例 4.2】实数绝对值的计算。

计算用户给定实数的绝对值。图 4.7 分别给出了该计算问题的 IPO 描述、流程图描述和 Python 代码描述。

【微实例 4.3】整数累加。

计算 1 到正整数 R 的算术和。图 4.8 分别给出了该计算问题的 IPO 描述、流程图描述和 Python 代码描述。

IPO 描述、流程图描述和 Python 代码描述是解决计算问题的 3 种描述方式，细致程度逐步递进。IPO 描述主要用于区分程序的输入输出关系，重点在于结构划分，算法主要采用自然语言描述。流程图描述侧重于描述算法的具体流程关系，流程图的结构化关系相比自然语言描述更进一步，有助于阐述算法的具体操作过程。Python 代码描述是最终的程序产出，最为细致。

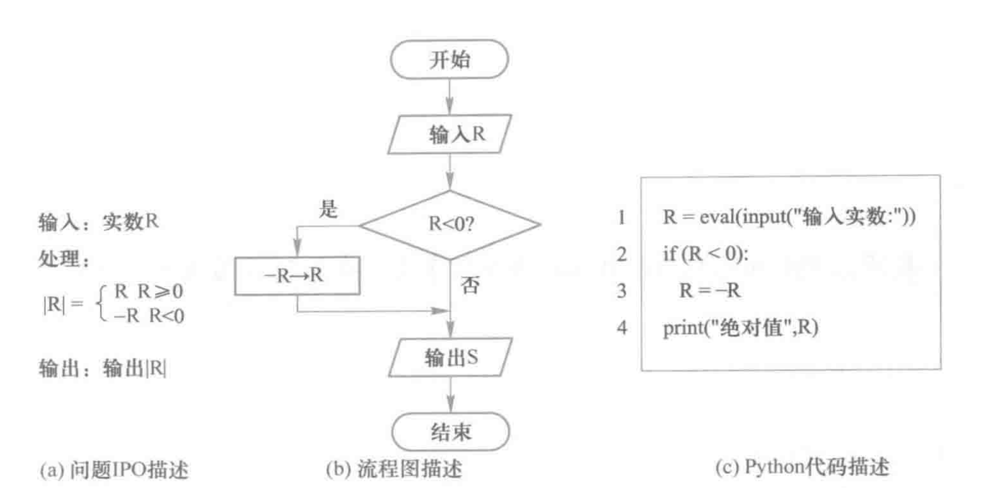


图 4.7　分支结构在三种描述下的对比

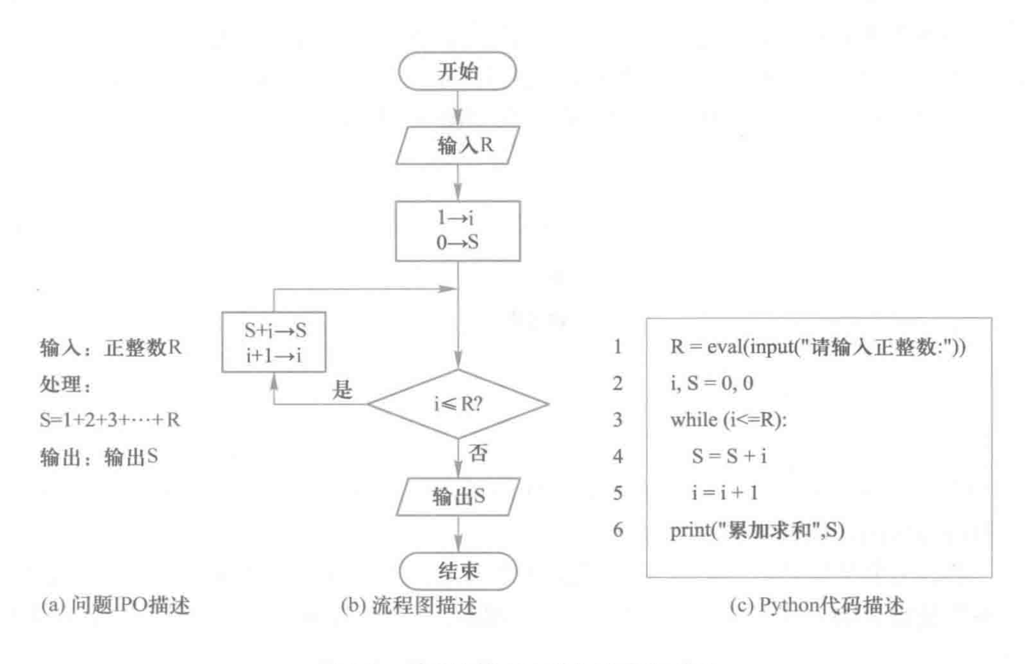


图 4.8　循环结构在三种描述下的对比

思考与练习

4.1　判断题：复杂的程序结构都是由基本结构组合而成。

4.2　判断题：分支结构可以向已经执行过的语句部分跳转（即向后跳转）。

4.3　下面是流程图的基本元素的是（　　）。

A. 判断框　　B. 顺序结构　　C. 分支结构　　D. 循环结构

4.4　循环结构可以使用 Python 语言中的（　　）语句实现？

A. print　　B. while　　C. loop　　D. if

4.2　程序的分支结构

要点：Python 通过 if、elif、else 等保留字提供单分支、二分支和多分支结构。

4.2.1　单分支结构：if 语句

Python 中 if 语句的语法格式如下：

```
if  <条件>:
    <语句块>
```

语句块是 if 条件满足后执行的一个或多个语句序列，语句块中语句通过与 if 所在行形成缩进表达包含关系。if 语句首先评估条件的结果值，如果结果为 True，则执行语句块中的语句序列，然后控制转向程序的下一条语句。如果结果为 False，语句块中的语句会被跳过。if 语句的控制过程如图 4.9 所示。

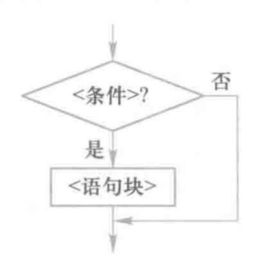


图 4.9　if 语句的控制流程图

if 语句中语句块执行与否依赖于条件判断。但无论什么情况，控制都会转到 if 语句后与该语句同级别的下一条语句。

if 语句中条件部分可以使用任何能够产生 True 或 False 的语句或函数。形成判断条件最常见的方式是采用关系操作符。Python 语言共有 6 个关系操作符，如表 4.1 所示。

表 4.1　Python 的关系操作符（共 6 个）

操　作　符	数　学　符　号	操作符含义
<	$<$	小于
<=	$\leqslant$	小于或等于
>=	$\geqslant$	大于或等于
>	$>$	大于
==	$=$	等于
!=	$\neq$	不等于

特别注意，Python 使用“=”表示赋值语句，使用“==”表示等于。

【微实例 4.4】PM 2.5 空气质量提醒（1）。

空气污染是当下社会比较关注的问题，PM2.5 是衡量空气污染的重要指标。PM2.5 是指大气中直径小于或等于 2.5 μm 的可入肺颗粒物。PM2.5 颗粒粒径小，含大量有毒、有害物质且在大气中停留时间长、输送距离远，因而对人体健康和大气环境质量有很大影响。目前空气质量等级以 PM2.5 数值划分为 6 级。PM2.5 数值在 0～35 空气质量为优，35～75 为良，75～115 为轻度污染，115～150 为中度污染，150～250 为重度污染，250～500 为严重污染。

一个简化版的空气质量标准采用三级模式：0～35 为优，35～75 为良，75 以上为污染。人们也许不关心 PM2.5 指数值具体为多少，而更关心空气质量到底怎样。计算机可以通过 PM2.5 指数分级发布空气质量提醒。该问题的 IPO 描述如下。

输入：接收外部输入的 PM2.5 值

处理：

　　if PM2.5 值 ≥75，打印空气污染警告
　　if 35≤ PM2.5 值 <75，打印空气质量良，建议适度户外运动
　　if PM2.5 值 <35，打印空气质量优，建议户外运动

输出：打印空气质量提醒

微实例 4.4 的完整代码如下：

微实例 4.4　　m4.4PM25Warning.py

```
1  PM = eval(input("请输入 PM2.5 数值: "))
2  if 0<= PM < 35:
3      print("空气优质，快去户外运动!")
4  if 35 <= PM <75:
5      print("空气良好，适度户外活动! ")
6  if 75 <= PM:
7      print("空气污染，请小心! ")
```

源代码 4-1：PM 2.5 空气质量提醒（1）



微实例 4.4 展示了用数字进行条件比较的例子，字符或字符串也可以用于条件比较。字符串比较本质上是字符串对应 Unicode 编码的比较，因此，字符串的比较按照字典顺序进行。例如，英文大写字符对应的 Unicode 编码比小写字符小。以下是一些例子：

```
>>>4 < 5
True
>>>"python" == "python"
True
>>>"Python" > "python"
False
```

4.2.2 二分支结构：if-else 语句

Python 中 if-else 语句用来形成二分支结构，语法格式如下：

```
if  <条件>:
    <语句块 1>
else:
    <语句块 2>
```

语句块 1 是在 if 条件满足后执行的一个或多个语句序列，语句块 2 是 if 条件不满足后执行的语句序列。二分支语句用于区分条件的两种可能，即 True 或者 False，分别形成执行路径。

【微实例 4.5】PM 2.5 空气质量提醒（2）。

如果用户只关心空气质量是否污染两种情况，可以通过二分支语句完成。

微实例 4.5　　　　m4.5PM25Warning.py

```
1  PM = eval(input("请输入 PM2.5 数值: "))
2  if PM >= 75:
3      print("空气存在污染，请小心！")
4  else:
5      print("空气没有污染，可以开展户外运动!")
```

源代码 4-2：
PM 2.5 空气质量提醒（2）



二分支结构还有一种更简洁的表达方式，适合通过判断返回特定值，语法格式如下：

```
<表达式 1>  if  <条件> else <表达式 2>
```

其中，表达式 1/2 一般是数字类型或字符串类型的一个值，微实例 4.5 可以改造为

```
1  PM = eval(input("请输入 PM2.5 数值: "))
2  print("空气{}污染!".format("存在" if PM >= 75 else "没有"))
```

if-else 的紧凑结构非常适合对特殊值处理的情况，其他例子如下：

```
>>>count = 2
>>>count if count!=0 else "不存在"
2
>>>count = 0
>>>count if count!=0 else "不存在"
"不存在"
```

拓展： 通往天堂的选择

> 你的面前有两扇门，一扇通往天堂，一扇通往地狱。每扇门前各站着一个人，两个人中一个人说真话，一个人说假话，现在你只能选择问其中一个人一个问题，你该怎样问才能确定通往天堂的大门呢？

4.2.3　多分支结构：if-elif-else 语句

Python 的 if-elif-else 描述多分支结构，语句格式如下，控制流程图如图 4.10 所示。

```
if <条件 1>:
    <语句块 1>
elif <条件 2>:
    <语句块 2>
...
else:
    <语句块 N>
```

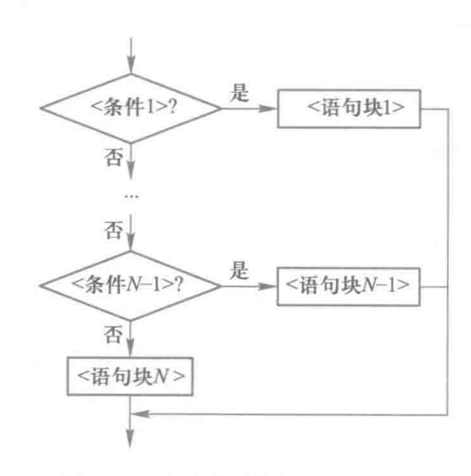


图 4.10　多分支结构的控制流程图

多分支结构是二分支结构的扩展，这种形式通常用于设置同一个判断条件的多条执行路径。Python 依次评估寻找第一个结果为 True 的条件，执行该条件下的语句块，结束后跳过整个 if-elif-else 结构，执行后面的语句。如果没有任何条件成立，else 下面的语句块将被执行。else 子句是可选的。

微实例 4.4 通过多条独立的 if 语句对同一个变量 PM 进行判断，这种情况更适合多分支结构，改造后的代码如下：

源代码 4-3：PM 2.5 空气质量提醒（3）



```
1  PM = eval(input("请输入 PM2.5 数值: "))
2  if 0<= PM < 35:
3      print("空气优质，快去户外运动!")
4  elif 35 <= PM <75:
5      print("空气良好，适度户外活动! ")
```

```
else:
    print("空气污染，请小心！")
```

思考与练习

4.5　判断题：简单分支结构是最基础的程序结构，在设计中一般用不到。

4.6　判断题：多分支结构是使用最广泛的结构，可替代任何选择性结构。

4.7　判断题：Python 语法认为条件 x<=y<=z 是合法的。

4.8　Python 通过（　　）来判断操作是否在分支结构中。

A．括号　　B．缩进　　C．花括号　　D．冒号

4.9　请分析下面的程序，若输入 score 为 80，输出 grade 为多少？是否符合逻辑？为什么？

```
if score >= 60.0:
    grade = 'D'
elif score >= 70.0:
    grade = 'C'
elif score >= 80.0:
    grade = 'B'
else score >= 90.0:
    grade = 'A'
```

4.3　实例 5：身体质量指数 BMI

> **要点：** 这是一个计算 BMI 并通过分支结构给出身体质量分类的实例。

改革开放近 40 年，中国取得了世界瞩目的发展成就，人民生活水平显著提高，越来越多人开始关注“身体质量”，其中，肥胖程度最受关注。身体质量指数（Body Mass Index，BMI）是国际上常用的衡量人体肥胖程度和是否健康的重要标准，主要用于统计分析。肥胖程度的判断不能采用体重的绝对值，它天然与身高有关。因此，BMI 通过人体体重和身高两个数值获得相对客观的参数，并用这个参数所处范围衡量身体质量。

BMI 的定义如下：

$$BMI = 体重（kg）/身高^2（m^2）$$

例如，一个人身高 1.75 m、体重 75 kg，他的 BMI 值为 24.49。

BMI 值可以客观地衡量人的肥胖程度或健康程度。世界卫生组织（World Health Organization，WHO）根据全球人口体重统计认为，BMI 值低于 18.5 kg/m^2 时“过

轻”，表明个体可能营养不良或者饮食无法保障；BMI 值高于 25 kg/m^2 时“过重”。我国卫生部也根据中国人体质给出了国内 BMI 参考值。更多 BMI 衡量标准如表 4.2 所示。

表 4.2　BMI 指标分类

分类	国际 BMI 值（kg/m^2）	国内 BMI 值（kg/m^2）
偏瘦	< 18.5	< 18.5
正常	18.5 ~ 25	18.5 ~ 24
偏胖	25 ~ 30	24 ~ 28
肥胖	≥30	≥28

拓展： 中国居民膳食指南

《中国居民膳食指南（2016）》是 2016 年 5 月 13 日由国家卫生计生委疾控局发布的指导中国居民饮食的权威资料。该指南针对 2 岁以上的所有健康人群提出 6 条核心推荐，分别为：食物多样，谷类为主；吃动平衡，健康体重；多吃水果、奶类、大豆；适量吃鱼、禽、蛋、瘦肉；少盐少油，控糖限酒；杜绝浪费，新兴食尚。

指南建议平均每天摄入 12 种以上食物，每周 25 种以上。各年龄段人群都应坚持日常身体活动，每周至少进行 5 天中等强度身体活动，累计 150 分钟以上。蔬菜水果是平衡膳食的重要组成部分，吃各种各样的奶制品，经常吃豆制品，适量吃坚果。鱼、禽、蛋和瘦肉摄入要适量。少吃肥肉、烟熏和腌制肉食品。成人每天食盐不超过 6 g，每天烹调油 25 ~ 30 g。足量饮水，成年人每天 7 ~ 8 杯，约 1 500 ~ 1 700 ml，提倡饮用白开水和茶水。健康饮食，你做到了哪些？

本实例编写一个根据体重和身高计算 BMI 值的程序，同时输出国际和国内的 BMI 指标建议值。该问题的 IPO 描述如下。

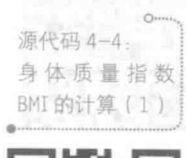
输入：身高和体重值

处理：计算 BMI 值，并根据 BMI 指标分类找到合适类别

输出：打印指标分类信息

该实例的完整代码如下，请注意各判断条件及后面的注释。其中第 2 行最后采用反斜杠（\）将很长的一行分解为两行书写，对于 Python 解释来说，这是一行代码。

实例代码 5.1　　e5.1CalBMI.py





```
1  #e5.1CalBMI.py
2  height, weight = eval(input("请输入身高(米)和体重\
                      (公斤)[逗号隔开]: "))
3  bmi = weight / pow(height, 2)
4  print("BMI 数值为: {:.2f}".format(bmi))
5  who, dom = "", ""
6  if bmi < 18.5:   # WHO 标准
7      who = "偏瘦"
```

```
8   elif bmi < 25:   # 18.5 <= bmi < 25
9       who = "正常"
10  elif bmi < 30:   # 25 <= bmi < 30
11      who = "偏胖"
12  else:
13      who = "肥胖"
14  if bmi < 18.5:    # 我国卫生部标准
15      dom = "偏瘦"
16  elif bmi < 24:    # 18.5 <= bmi < 24
17      dom = "正常"
18  elif bmi < 28:    # 24 <= bmi < 28
19      dom = "偏胖"
20  else:
21      dom = "肥胖"
22  print("BMI 指标为:国际'{0}', 国内'{1}'".format(wto, dom))
```

程序执行后的效果如下：

```
>>>
请输入身高(米)和体重(公斤)[逗号隔开]: 1.75, 75
BMI 数值为: 24.49
BMI 指标为:国际'正常', 国内'偏胖
```

实例代码 5.1 采用了多分支结构对 BMI 数值按照不同区间范围进行分类，这种采用 if-elif-else 分支语句进行程序设计的方式十分常见。对于需要同时打印国际和国内两套 BMI 标准，程序采用两个 if-elif-else 语句分别计算两类不同 BMI 值。这种做法的好处是代码清晰明了，容易调试。实例代码 5.2 将两套指标合成一个 if-elif-else 语句实现。

源代码 4-5：身体质量指数 BMI 的计算（2）

实例代码 5.2　　　　e5.2CalBMI.py

```
1   #e5.2CalBMI.py
2   height, weight = eval(input("请输入身高(米)和体重\
                            (公斤)[逗号隔开]: "))
3   bmi = weight / pow(height, 2)
4   print("BMI 数值为: {:.2f}".format(bmi))
5   who, dom = "", ""
6   if bmi < 18.5:
7       who, dom = "偏瘦", "偏瘦"
8   elif 18.5 <= bmi < 24:
9       who, dom = "正常", "正常"
10  elif 24 <= bmi < 25:
11      who, dom = "正常", "偏胖"
12  elif 25 <= bmi < 28:
```

```
13        who, dom = "偏胖", "偏胖"
14    elif 28 <= bmi < 30:
15        who, dom = "偏胖", "肥胖"
16    else:
17        who, dom = "肥胖", "肥胖"
18    print("BMI 指标为:国际'{0}', 国内'{1}'".format(who, dom))
```

比较实例代码 5.1 和实例代码 5.2 可以看到，if 语句的运用主要与程序编写者对问题的理解及算法设计有关，采用一组 if 语句将两套 BMI 指标融合在一起，这实际上是算法的改变。即使对专业程序员来说，程序的简洁性和可读性都比更少的代码行数重要，这里，推荐采用实例代码 5.1 的方式编写程序。

思考与练习

4.10　观察实例代码 5.1 中的第 8 行和第 10 行，思考为何代码不按照注释方式写全变量的最小边界。

4.11　判断题：Python 中条件 24 <= 28 < 25 是合法的，且输出为 False。

4.12　实例代码 5.1 中第 2 行最后的反斜杠（\）有什么作用？

4.4　程序的循环结构

> **要点：** Python 通过 for、while 等保留字提供遍历循环和无限循环的结构。

根据循环执行次数的确定性，循环可以分为确定次数循环和非确定次数循环。确定次数循环指循环体对循环次数有明确的定义，这类循环在 Python 中被称为“遍历循环”，其中，循环次数采用遍历结构中的元素个数来体现，具体采用 for 语句实现。非确定次数循环指程序不确定循环体可能的执行次数，而通过条件判断是否继续执行循环体，Python 提供了根据判断条件执行程序的无限循环，采用 while 语句实现。

4.4.1　遍历循环：for 语句

Python 通过保留字 for 实现“遍历循环”，基本使用方法如下：

```
for <循环变量> in <遍历结构>:
    <语句块>
```

之所以称为“遍历循环”，是因为 for 语句的循环执行次数是根据遍历结构中元素个数确定的。遍历循环可以理解为从遍历结构中逐一提取元素，放在循环变量中，对于所提取的每个元素执行一次语句块。

遍历结构可以是字符串、文件、组合数据类型或 range()函数等，常用的使用方式如下：

```
循环 N 次                遍历文件 fi 的每一行      遍历字符串 s         遍历列表 ls
for i in range(N):       for line in fi:          for c in s:          for item in ls:
    <语句块>                 <语句块>                 <语句块>             <语句块>
```

遍历循环还有一种扩展模式，使用方法如下：

```
for <循环变量> in <遍历结构>:
    <语句块 1>
else:
    <语句块 2>
```

在这种扩展模式中，当 for 循环正常执行之后，程序会继续执行 else 语句中的内容。else 语句只在循环正常执行并结束后才执行，因此，可以在<语句块 2>中放置判断循环执行情况的语句。4.4.3 节将结合 continue 和 break 语句进一步讲解 for 语句中 else 的用法。这里先给出一个小例子：

```
1  for s in "BIT":
2      print("循环进行中： " + s)
3  else:
4      s = "循环正常结束"
5  print(s)
```

程序执行后的结果如下：

```
>>>
循环进行中： B
循环进行中： I
循环进行中： T
循环正常结束
```

4.4.2 无限循环：while 语句

很多应用无法在执行之初确定遍历结构，这需要编程语言提供根据条件进行循环的语法，称为无限循环，又称条件循环。无限循环一直保持循环操作直到循环条件不满足才结束，不需要提前确定循环次数。

Python 通过保留字 while 实现无限循环，基本使用方法如下：

```
while <条件>:
    <语句块>
```

其中条件与 if 语句中的判断条件一样，结果为 True 和 False。

while 语义很简单，当条件判断为 True 时，循环体重复执行语句块中语句；当条件为 False 时，循环终止，执行与 while 同级别缩进的后续语句。

无限循环也有一种使用保留字 else 的扩展模式，使用方法如下：

```
while  <条件>:
    <语句块 1>
else:
    <语句块 2>
```

在这种扩展模式中，当 while 循环正常执行后，程序会继续执行 else 语句中的内容。else 语句只在循环正常执行后才执行，因此，可以在语句块 2 中放置判断循环执行情况的语句，例如：

```
1    s, idx = "BIT", 0
2    while idx < len(s):
3        print("循环进行中：" + s[idx])
4        idx += 1
5    else:
6        s = "循环正常结束"
7    print(s)
```

程序执行后的结果如下：

```
>>>
循环进行中：B
循环进行中：I
循环进行中：T
循环正常结束
```

如果通过 while 实现一个计数循环，需要在循环之前对计数器 idx 进行初始化，并在每次循环中对计数器 idx 进行累加，如上述代码第 4 行。对比一下，在 for 循环中循环变量逐一取自遍历结构，不需要程序维护计数器。

拓展： 科幻电影中的循环故事

> 循环不仅是编程语言的组成部分，也是好莱坞科幻题材的最爱。在编剧笔下，情节在时间轮回与空间虚实中无限循环，剧情展示着主人公享受并发现循环奥秘的历程，十分刺激好看！下面推荐几部无限循环类电影，感受一下控制结构的奥秘吧！例如，《恐怖游轮》、《源代码》、《蝴蝶效应》、《罗拉快跑》、《明日边缘》、《土拨鼠之日》等。

4.4.3　循环保留字：break 和 continue

循环结构有两个保留字：break 和 continue，它们用来辅助控制循环执行。

break 用来跳出最内层 for 或 while 循环，脱离该循环后程序从循环代码后继续执行，例如：

```
1  for s in "BIT":
2      for i in range(10):
3          print(s, end="")
4          if s=="I":
5              break
```

程序执行后的结果如下:

```
>>>
BBBBBBBBBBITTTTTTTTTT
```

其中,break 语句跳出了最内层 for 循环,但仍然继续执行外层循环。每个 break 语句只有能力跳出当前层次循环。

continue 用来结束当前当次循环,即跳出循环体中下面尚未执行的语句,但不跳出当前循环。对于 while 循环,继续求解循环条件。而对于 for 循环,程序流程接着遍历循环列表。对比 continue 和 break 语句,如下:

```
1  for s in "PYTHON":
2      if s=="T":
3          continue
4      print(s, end="")
```

```
1  for s in "PYTHON":
2      if s=="T":
3          break
4      print(s, end="")
```

两个程序执行后的结果分别如下:

```
>>>
PYHON
```

```
>>>
PY
```

continue 语句和 break 语句的区别是,continue 语句只结束本次循环,而不终止整个循环的执行;而 break 语句则是结束整个循环过程,不再判断执行循环的条件是否成立。

for 循环和 while 循环中都存在一个 else 扩展用法。else 中的语句块只在一种条件下执行,即循环正常遍历了所有内容或由于条件不成立而结束循环,没有因为 break 或 return(函数返回中使用的保留字)而退出。continue 保留字对 else 没有影响。看下面两个例子:

```
1  for s in "PYTHON":
2      if s=="T":
3          continue
4      print(s, end="")
5  else:
6      print("正常退出")
```

```
1  for s in "PYTHON":
2      if s=="T":
3          break
4      print(s, end="")
5  else:
6      print("正常退出")
```

两个程序执行后的结果分别如下:

```
>>>
PYHON 正常退出
```

```
>>>
PY
```

阶段测试 4-1：Python 程序控制结构小测验

思考与练习

4.13　判断题：所有 for 循环语句都可以用 while 循环语句改写。

4.14　判断题：while 循环只能用来实现无限循环的编程。

4.15　判断题：死循环对编程没有任何益处。

4.16　下面不是 while 循环的特点的是（　　）。

A．提高程序的复用性　　B．能够实现无限循环

C．如果不小心会出现死循环　　D．必须提供循环的次数

4.17　以下可以终结一个循环的保留字是（　　）。

A．if　　B．break　　C．exit　　D．continue

程序练习 4-1：半小时学 Python 控制结构



4.5　模块 2：random 库的使用

要点： 随机运算的标准函数库 random 共提供 9 个常用函数。

4.5.1　random 库概述

随机数在计算机应用中十分常见，Python 内置的 random 库主要用于产生各种分布的伪随机数序列。random 库采用梅森旋转算法（Mersenne Twister）生成伪随机数序列，可用于除随机性要求更高的加解密算法外的大多数工程应用。

使用 random 库的主要目的是生成随机数，因此，读者只需要查阅该库中随机数生成函数，找到符合使用场景的函数即可。该库提供了不同类型的随机数函数，所有函数都是基于最基本的 random.random() 函数扩展实现。

图片资料 4-1：Python 快速参考之 random、jieba、datetime 库



拓展： 伪随机数和真随机数

随机数或随机事件是不确定性的产物，其结果是不可预测、产生之前不可预见。无论计算机产生的随机数看起来多么“随机”，它们也不是真正意义上的随机数。因为计算机是按照一定算法产生随机数的，其结果是确定的、可预见的，称为“伪随机数”。真正意义上的随机数不能评价。如果存在评价随机数的方法，即判断一个数是否是随机数，那么这个随机数就有确定性，将不再是随机数。

4.5.2　random 库解析

表 4.3 列出了 random 库常用的 9 个随机数生成函数。

表 4.3 random 库的常用函数（共 9 个）

函数	描述
seed(a=None)	初始化随机数种子，默认值为当前系统时间
random()	生成一个[0.0, 1.0)之间的随机小数
randint(a, b)	生成一个[a,b]之间的整数
getrandbits(k)	生成一个 k 比特长度的随机整数
randrange(start, stop[, step])	生成一个[start, stop)之间以 step 为步数的随机整数
uniform(a, b)	生成一个[a, b]之间的随机小数
choice(seq)	从序列类型，例如列表中随机返回一个元素
shuffle(seq)	将序列类型中的元素随机排列，返回打乱后的序列
sample(pop, k)	从 pop 类型中随机选取 k 个元素，以列表类型返回

random 库的引用方法与 math 库一样，可以采用下面两种方式实现：

```
import random
```

或

```
from random import *
```

使用 random 库的一些例子如下，请读者注意，这些语句每次执行后的结果不一定一样：

```
>>>from random import*
>>>random()
0.2922089114412476
>>>uniform(1,10)
1.5913082783598524
>>>uniform(1,20)
7
>>>randrange(0,100,4) #从 0 开始到 100 以 4 递增的元素中随机返回
96
>>>choice(range(100))
97
>>>ls = list(range(10))
>>>print(ls)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>shuffle(ls)
>>>print(ls)
[5, 8, 4, 7, 6, 9, 3, 0, 2, 1]
```

生成随机数之前可以通过 seed()函数指定随机数种子，随机数种子一般是一个整数，只要种子相同，每次生成的随机数序列也相同。这种情况便于测试和同步数据，例如：

```
>>>seed(125) # 随机数种子赋值 125
>>>"{}.{}.{}".format(randint(1,10),randint(1,10),randint(1,10))
'4.4.10'
```

```
>>>"{}.{}.{}".format(randint(1,10),randint(1,10),randint(1,10))
'5.10.3'
>>>seed(125) # 再次给随机数种子赋值 125
>>>"{}.{}.{}".format(randint(1,10),randint(1,10),randint(1,10))
'4.4.10'
```

由上述语句可以看出，在设定相同种子后，每次调用随机函数生成的随机数是相同的。这是随机数种子的作用，也是伪随机序列的应用之一。

思考与练习

程序练习 4-2：十分钟学 random 库



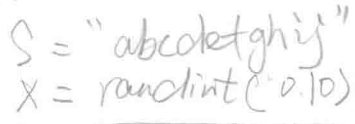
4.18　从 random 库中选取相应的函数满足下列条件。

（1）随机生成 100 内的 10 个整数。

（2）随机选取 0 到 100 间的奇数。

（3）从字符串'abcdefghij'中随机选取 4 个字符。

（4）随机选取列表['apple', 'pear', 'peach', 'orange']中的 1 个字符串。



4.6　实例 6：π 的计算

> **要点：** 这是一个采用蒙特卡罗方法计算圆周率的实例。

π（圆周率）是数学和物理学普遍存在的常数之一，它定义了一个标准圆周长与直径之比。众所周知，π 是一个无理数，即无限不循环小数。精确求解圆周率 π 是几何学、物理学和很多工程学科的关键。

对 π 的精确求解曾经是数学历史上一直难以解决的问题之一，因为 π 无法用任何精确公式表示，在电子计算机出现以前，π 只能通过一些近似公式的求解得到，直到 1948 年，人类才以人工计算方式得到 π 的 808 位精确小数。

迄今为止求解圆周率最好的方法是利用 BBP 公式，该公式如下：

$$\pi=\sum_{k=0}^{\infty}\left[\frac{1}{16^k}\left(\frac{4}{8k+1}-\frac{2}{8k+4}-\frac{1}{8k+5}-\frac{1}{8k+6}\right)\right]$$

随着计算机的出现，数学家找到了求解 π 的另类方法：蒙特卡罗（Monte Carlo）方法，又称随机抽样或统计试验方法。该方法属于计算数学的一个分支，由于其能够真实地模拟实际物理过程，因此，解决问题与实际非常符合，可以得到很圆满的结果。蒙特卡罗方法广泛应用于数学、物理学和工程领域。

当所要求解的问题是某种事件出现的概率，或者是某个随机变量的期望值时，它们可以通过某种“试验”的方法，得到这种事件出现的频率，或者这个随机变数的平均值，并用它们作为问题的解。这是蒙特卡罗方法的基本思想。

应用蒙特卡罗方法求解 π 的基本步骤如下：随机向如图 4.11 所示的单位正方形和圆结构，抛洒大量“飞镖”点，计算每个点到圆心的距离从而判断该点在圆内或者圆外，用圆内的点数除以总点数就是 π/4 值。随机点数量越大，越充分覆盖整个图形，计算得到的 π 值越精确。实际上，这个方法的思想是利用离散点值表示图形的面积，通过面积比例来求解 π 值。

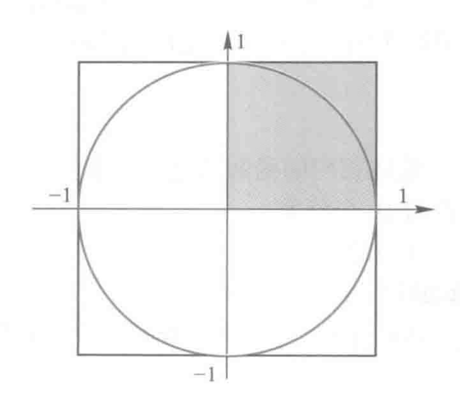


图 4.11　计算 π 使用的正方形和圆形

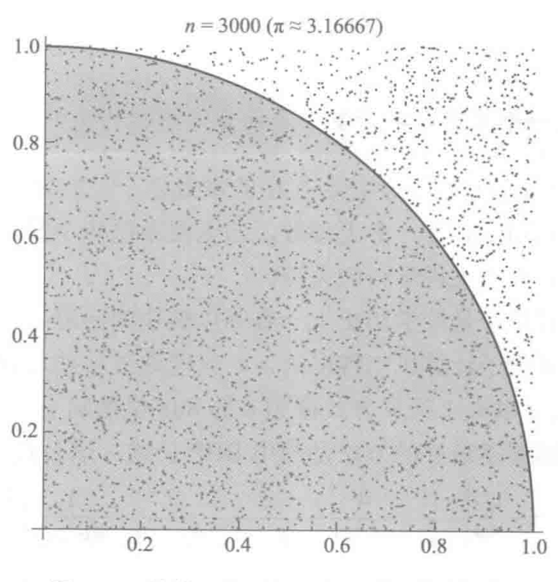


图 4.12　计算 π 使用的 1/4 区域和抛点过程

为了简化计算，一般利用图形的 1/4 求解 π 值，如图 4.12 所示。该问题的 IPO 表示如下。

输入：抛点数

处理：计算每个点到圆心的距离，统计在圆内点的数量

输出：π 值

采用蒙特卡罗方法求解 π 值的 Python 程序如下：

实例代码 6.1　　　　　　e6.1CalPi.py

源代码 4-6：蒙特卡罗方法解 π 值

```
1   #e6.1CalPi.py
2   from random import random
3   from math import sqrt
4   from time import clock
5   DARTS = 10000
6   hits = 0.0
7   clock()
8   for i in range(1, DARTS+1):
9       x, y = random(), random()
10      dist = sqrt(x ** 2 + y ** 2)
11      if dist <= 1.0:
12          hits = hits + 1
13  pi = 4 * (hits/DARTS)
14  print("Pi 值是{}.".format(pi))
15  print("运行时间是: {:5.5}s".format(clock()))
```

上述代码中，random()函数随机返回一个在[0,1)之间的浮点数，用两个随机数给出随机抛点(x,y)的坐标。sqrt()函数来自于数学库 math，用来求解输入数据的平方根。第一次调用 clock()函数启动一个新的计时器，第二次调动 clock()函数返回启动计时器后的时间。代码中 DARTS 表示抛点数，初始设定为 1 000。该程序运行结果如下：

```
>>>
Pi 值是 3.144.
运行时间是: 0.016477s
```

计算得到的 π 值为 3.144，与大家熟知的 3.141 5 相差较远，原因是 DARTS 点数量较少，无法更精确刻画面积的比例关系。表 4.4 列出了不同 DARTS 值情况下该程序的运行情况。

表 4.4　不同抛点数产生的精度和运行时间

DARTS	π	运 行 时 间
2^{10}	3.109 375	0.011 s
2^{11}	3.138 671	0.012 s
2^{12}	3.150 390	0.014 s
2^{13}	3.143 554	0.018 s
2^{14}	3.141 357	0.030 s
2^{15}	3.147 827	0.049 s
2^{16}	3.141 967	0.116 s
2^{18}	3.144 577	0.363 s
2^{20}	3.142 669 677 7	1.255 s
2^{25}	3.141 697 883 6	40.13 s

可以看到，随着 DARTS 数量的增加，当达到 2^{20} 数量级时，π 的值就相对准确了。进一步增加 DARTS 数量，能够进一步增加 π 的精度。

本节以 π 的计算为例，重点讲解蒙特卡罗方法，希望读者能够将该方法运用到其他工程问题中。当然，求解 π 可以使用 BBP 公式，请读者根据本节开始给出的公式编写代码，用另一种方法获得 π 的值。

拓展： 圆周率日

圆周率日是一年一度庆祝数学常数 π 的节日，由大学倡导，在一些大学数学系会有庆祝活动。圆周率日在每年的 3 月 14 日，通常庆祝活动在下午 1 时 59 分开始，象征圆周率的 6 位近似值 3.141 59，有时甚至精确到 26 秒开始，以象征圆周率的 8 位近似值 3.141 592 6。对于那些习惯 24 h 计时的人，可以在 3 月 14 日凌晨 1 时 59 分或者下午 3 时 9 分（15 时 9 分）开始庆祝。

——竟然还有圆周率日！有没有自然对数日？有没有无作业日？

——只要人类还有创造力，什么特殊日子都可能有。

思考与练习

4.19 请修改代码，随机生成更多的点，观察 π 的值是否更加精确。

4.20 怎样让 π 计算的程序每次运行结果都一样？

4.21 请调研一下，还有哪些计算问题可以用蒙特卡罗方法求解？

4.7 程序的异常处理

要点： Python 通过 try、except 等保留字提供异常处理功能。

4.7.1 异常处理：try-except 语句

观察下面这段小程序：

```
1  num = eval(input("请输入一个整数: "))
2  print(num**2)
```

当用户输入数字时，程序正常执行，如果用户输入的不是数字呢？

```
>>>
请输入一个整数: 100
10000
>>>
```

```
请输入一个整数: NO
Traceback (most recent call last):
  File "D:/PythonPL/echoInt.py", line 1, in <module>
    num = eval(input("请输入一个整数: "))
  File "<string>", line 1, in <module>
NameError: name 'No' is not defined
```

可以看到，Python 解释器返回了异常信息，同时退出程序，图 4.13 具体说明了这个异常信息中各部分的含义。

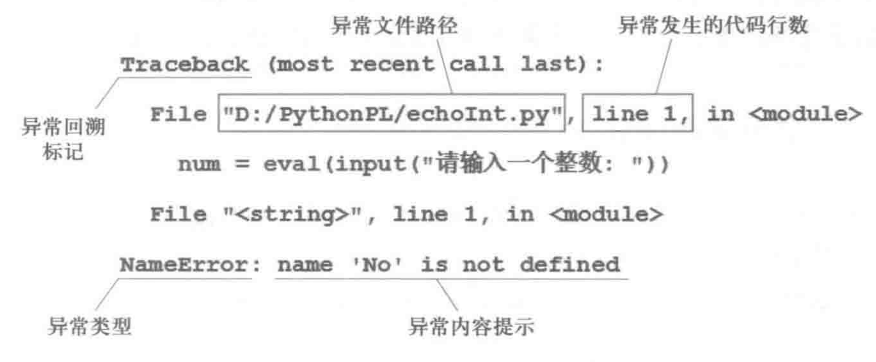


图 4.13　Python 异常信息含义说明

Python 异常信息中最重要的部分是异常类型，它表明发生异常的原因，也是程序处理异常的依据。

Python 使用 try-except 语句实现异常处理，其基本语法格式如下：

```
try:
    <语句块 1>
except <异常类型>:
    <语句块 2>
```

语句块 1 是正常执行的程序内容，当发生异常时执行 except 保留字后面的语句块，为上述小程序增加异常处理，代码如下：

源代码 4-7：异常处理的小例子（1）

```
1  try:
2      num = eval(input("请输入一个整数: "))
3      print(num**2)
4  except NameError:
5      print("输入错误，请输入一个整数!")
```

该程序执行结果如下：

```
>>>
请输入一个整数: NO
输入错误，请输入一个整数!
```

拓展：异常和错误

编程语言的异常和错误是两个相似但不同的概念。异常和错误都可能引起程序执行错误而退出，它们属于程序没有考虑到的例外情况（exception）。然而，绝大多数不可控因素是可以预见的，例如，程序期望获得数字输入却得到了其他字符输入、打开一个不存在的文件等。这种可以预见的例外情况称为“异常”(checked exception)，异常发生后经过妥善处理可以继续执行。另外一些因为程序编码逻辑产生的不可预见的例外情况称为“错误”（unchecked exception），错误发生后程序无法恢复执行，而且程序本不该处理这类可能的例外，例如，对于一个包含 5 个字符的字符串，程序去索引其中第 6 个元素，这种错误完全可以避免。

4.7.2 异常的高级用法

除了最基本的 try-except 用法，Python 异常还有一些略微高级的用法，这些方法在实际程序设计中也十分常用。

try-except 语句可以支持多个 except 语句，语法格式如下：

```
try:
    <语句块 1>
except <异常类型 1>:
    <语句块 2>
…
except <异常类型 N>:
    <语句块 N+1>
except:
    <语句块 N+2>
```

其中，第 1 到第 N 个 except 语句后面都指定了异常类型，说明这些 except 所包含的语句块只处理这些类型的异常。最后一个 except 语句没有指定任何类型，表示它对应的语句块可以处理所有其他异常。这个过程与 if-elif-else 语句类似，是分支结构的一种表达方式，例如如下代码：

源代码 4-8：异常处理的小例子（2）

```
1  try:
2      alp = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
3      idx = eval(input("请输入一个整数："))
4      print(alp[idx])
5  except NameError:
6      print("输入错误，请输入一个整数！")
7  except:
8      print("其他错误")
```

该程序将用户输入的数字作为索引从字符串 alp 中返回一个字符，当用户输入非整数字符时，except NameError 异常被捕获到，提示用户输入类型错误，当用户

输入数字不在 0 到 25 之间时，异常被 except 捕获，程序打印其他错误信息，执行过程和结果如下：

```
>>>
请输入一个整数：NO
输入错误，请输入一个整数！
>>>
请输入一个整数：100
其他错误
```

除了 try 和 except 保留字外，异常语句还可以与 else 和 finally 保留字配合使用，语法格式如下：

```
try:
    <语句块 1>
except <异常类型 1>:
    <语句块 2>
else:
    <语句块 3>
finally:
    <语句块 4>
```

此处的 else 语句与 for 循环和 while 循环中的 else 一样，当 try 中的语句块 1 正常执行结束且没有发生异常时，else 中的语句块 3 执行，可以看作是对 try 语句块正常执行后的一种追加处理。finally 语句块则不同，无论 try 中的语句块 1 是否发生异常，语句块 4 都会执行，可以将程序执行语句块 1 的一些收尾工作放在这里，例如，关闭、打开文件等。采用这些保留字的异常处理控制流过程如图 4.14 所示。

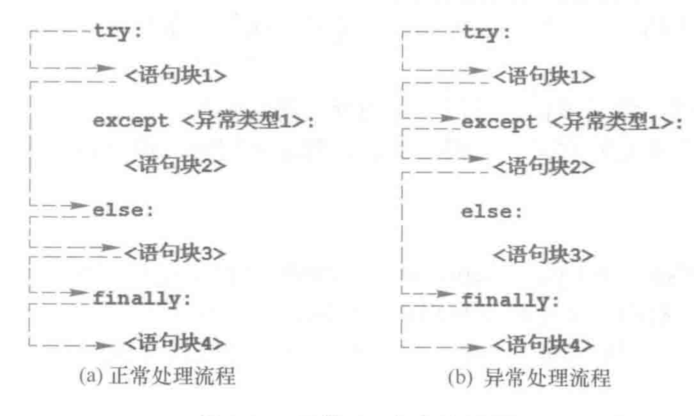
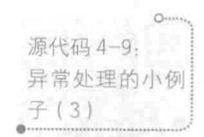


图 4.14　异常处理控制流过程





采用 else 和 finally 修改代码如下：

```
1  try:
2      alp = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
3      idx = eval(input("请输入一个整数："))
4      print(alp[idx])
```

```
5   except NameError:
6       print("输入错误，请输入一个整数！")
7   else:
8       print("没有发生异常")
9   finally:
10      print("程序执行完毕，不知道是否发生了异常")
```

执行过程和结果如下：

```
>>>
请输入一个整数：5
F
没有发生异常
程序执行完毕，不知道是否发生了异常
>>>
请输入一个整数：NO
输入错误，请输入一个整数！
程序执行完毕，不知道是否发生了异常
```

Python 能识别多种异常类型，但不建议读者编写程序时过度依赖 try-except 这种异常处理机制。try-except 异常一般只用来检测极少发生的情况，例如，用户输入的合规性或文件打开是否成功等。对于本节小例子中索引字符串超过范围的情况应该尽量在程序中采用 if 语句直接判断，而避免通过异常处理来应对这种可能发生的“错误”。

对于面向商业应用的软件产品，稳定性和可靠性是最重要的衡量指标之一。即使这类软件产品也不会滥用 try-except 类型语句。因为采用 try-except 语句会影响代码的可读性，增加代码维护难度，因此，一般只在关键地方采用 try-except 类型语句处理可能发生的异常。建议读者结合函数设计统筹应用异常处理。更多经验还需要实践来积累。

——没看懂，到底该什么时候该使用异常语句呢？

——程序员都是对它又爱又恨，其实，想用就用吧，用户体验好才是王道！

阶段测试 4-2：Python 异常处理小测验



思考与练习

4.22 请阐述一下 try、except、else、finally 保留字在异常处理中的作用。

4.23 如何利用异常处理机制判断用户输入的合规性？

4.24 如果不用异常处理机制，还有什么办法判断用户输入的合规性？

本 章 小 结

本章主要讲解程序的基本结构，包括分支结构和循环结构，介绍身体质量指数 BMI 的计算，用实例说明分支结构的使用。本章同时介绍了一个常用标准库 random

库，利用它实现了蒙特卡罗方法求解 π 的过程。最后介绍了程序的异常处理操作。

程序练习题

程序练习 4-3：
章节程序练习题



4.1　猜数游戏。在程序中预设一个 0~9 之间的整数，让用户通过键盘输入所猜的数，如果大于预设的数，显示“遗憾，太大了”；小于预设的数，显示“遗憾，太小了”，如此循环，直至猜中该数，显示“预测 N 次，你猜中了！”，其中 N 是用户输入数字的次数。

4.2　统计不同字符个数。用户从键盘输入一行字符，编写一个程序，统计并输出其中英文字符、数字、空格和其他字符的个数。

4.3　最大公约数计算。从键盘接收两个整数，编写程序求出这两个整数的最大公约数和最小公倍数（提示：求最大公约数可用辗转相除法，求最小公倍数则用两数的积除以最大公约数即可）。

4.4　猜数游戏续。改编程序练习题 4.1，让计算机能够随机产生一个预设数字，范围在 0～100 之间，其他游戏规则不变。

4.5　猜数游戏续。对于程序练习题 4.4 程序，当用户输入的不是整数（如字母、浮点数等）时，程序会终止执行退出。改编该程序，当用户输入出错时给出“输入内容必须为整数！”的提示，并让用户重新输入。

4.6　羊车门问题。有 3 扇关闭的门，一扇门后面停着汽车，其余门后是山羊，只有主持人知道每扇门后面是什么。参赛者可以选择一扇门，在开启它之前，主持人会开启另外一扇门，露出门后的山羊，然后允许参赛者更换自己的选择。请问：参赛者更换选择后能否增加猜中汽车的机会？——这是一个经典问题。

请使用 random 库对这个随机事件进行预测，分别输出参赛者改变选择和坚持选择获胜的机率。

4.7　请用异常处理改造实例 1，使其能够接收并处理用户的任何输入。

电子教案 5-1：
函数和代码复用

第 5 章　函数和代码复用

以代码行数来衡量程序设计的进度，就好比以重量来衡量飞机的制造进度。

Measuring programming progress by lines of code is like measuring aircraft building progress by weight.

——比尔·盖茨（Bill Gates）

微软公司创始人、软件工程师、慈善家

连续 22 年(1995—2016)蝉联《福布斯》全球富豪榜首富

学习目标

（1）掌握函数的定义和调用方法。
（2）理解函数的参数传递过程以及变量的作用范围。
（3）了解 lambda 函数。
（4）掌握时间日期标准库的使用。
（5）理解函数递归的定义和使用方法。

《关于一条连续而无切线,可由初等几何构作的曲线》是最早提到科赫曲线的书籍。科赫曲线是一种外形类似雪花的几何曲线,又称为雪花曲线。作为分形曲线的一种，它是数学的抽象，具有无限精细的结构。在科赫曲线基础上产生的科赫雪花更加美丽。

用程序绘制科赫曲线，探求雪花构成的极限奥秘，即刻开始！

5.1　函数的基本使用

> **要点：** 函数是一段具有特定功能的、可重用的语句组。

5.1.1　函数的定义

函数是一段具有特定功能的、可重用的语句组，用函数名来表示并通过函数名进行功能调用。函数也可以看作是一段具有名字的子程序，可以在需要的地方调用执行，不需要在每个执行的地方重复编写这些语句。每次使用函数可以提供不同的参数作为输入，以实现对不同数据的处理；函数执行后，还可以反馈相应的处理结果。

函数能够完成特定功能，与黑盒类似，对函数的使用不需要了解函数内部实现原理，只要了解函数的输入输出方式即可。严格地说，函数是一种功能抽象。

有些函数是用户自己编写的，称为自定义函数；Python 安装包也自带了一些函数和方法，包括 Python 内置的函数（如 abs()、eval()）、Python 标准库中的函数（如 math 库中的 sqrt()）等。

使用函数主要有两个目的：降低编程难度和代码重用。函数是一种功能抽象，利用它可以将一个复杂的大问题分解成一系列简单的小问题，然后将小问题继续划分成更小的问题，当问题细化到足够简单时，就可以分而治之，为每个小问题编写程序，并通过函数封装，当各个小问题都解决了，大问题也就迎刃而解。这是一种自顶向下的程序设计思想，8.3 节将详细介绍这种设计思想。函数可以在一个程序中的多个位置使用，也可以用于多个程序，当需要修改代码时，只需要在函数中修改一次，所有调用位置的功能都更新了，这种代码重用降低了代码行数和代码维护难度。

Python 使用 def 保留字定义一个函数，语法形式如下：

```
def <函数名>(<参数列表>):
    <函数体>
    return <返回值列表>
```

函数名可以是任何有效的 Python 标识符；参数列表是调用该函数时传递给它的值，可以有零个、一个或多个，当传递多个参数时各参数由逗号分隔，当没有参数时也要保留圆括号。函数定义中参数列表里面的参数是形式参数，简称为“形参”。函数体是函数每次被调用时执行的代码，由一行或多行语句组成。当需要返回值时，使用保留字 return 和返回值列表，否则函数可以没有 return 语句，在函数体结束位置将控制权返回给调用者。

函数调用和执行的一般形式如下：

<函数名>(<参数列表>)

此时，参数列表中给出要传入函数内部的参数，这类参数称为实际参数，简称为“实参”。

【微实例 5.1】生日歌。

过生日时要为朋友唱生日歌，歌词为

Happy birthday to you!

Happy birthday to you!

Happy birthday, dear <名字>

Happy birthday to you!

编写程序为 Mike 和 Lily 输出生日歌。最简单的实现方法是重复使用 print()语句，对 Mike 的生日歌输出如下：

```
1  print("Happy birthday to you!")
2  print("Happy birthday to you!")
3  print("Happy birthday, dear Mike!")
4  print("Happy birthday to you!")
```

其中，第 1、2、4 行代码相同，假如需要将 birthday 改为 new year，则每处都要修改。为了避免这种情况，可以用函数进行封装。上述代码中除第 3 行有微小不同外其余代码完全一致，这会带来重复代码，为了能够复用语句，考虑将代码修改为：

源代码 5-1：生日歌



微实例 5.1 m5.1HappyBirthday.py

```
1   def happy():
2       print("Happy birthday to you!")
3   def happyB(name):
4       happy()
5       happy()
6       print("Happy birthday, dear {}!".format(name))
7       happy()
8   happyB("Mike")
9   print()
10  happyB("Lily")
```

该程序输出结果如下：

```
>>>
Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Mike!
```

```
Happy birthday to you!

Happy birthday to you!
Happy birthday to you!
Happy birthday, dear Lily!
Happy birthday to you!
```

微实例 5.1 代码中第 3 行定义了一个函数 happyB()，括号中的 name 是形参，用来指代要输入函数的实际变量，并参与完成函数内部功能。第 8 和第 10 行调用两次 happyB()函数，输入的"Mike"和"Lily"是实参，替换 name，用于函数执行。

5.1.2 函数的调用过程

程序调用一个函数需要执行以下 4 个步骤。

（1）调用程序在调用处暂停执行。

（2）在调用时将实参复制给函数的形参。

（3）执行函数体语句。

（4）函数调用结束给出返回值，程序回到调用前的暂停处继续执行。

对微实例 5.1 的生日歌程序跟踪分析。第 1 到第 7 行是函数定义，函数只有在被调用时才执行，因此，前 7 行代码不直接执行。程序最先执行的语句是第 8 行的 happyB("Mike")。当 Python 执行到这行时，由于调用了 happyB()函数，当前执行暂停，程序用实参"Mike"替换 happyB(name)中的形参 name，形参被赋值为实参的值，类似执行了如下语句：

```
name = "Mike"
```

然后，使用实参代替形参执行函数体内容。当函数执行完毕后，重新回到第 8 行，继续执行余下语句。函数第 8 行的执行过程如图 5.1 所示，这里函数 happyB()的变量 name 被自动替换为"Mike"。

```
                               name="Mike"
happyB("Mike") ———→  def happyB(name):
print()                    happy()
happyB("Lily")             happy()
                           print("Happy birthday, dear!".format(name))
                           happy()
```

图 5.1 微实例 5.1 中 happyB()的被调用过程

当程序执行 happyB()函数体时，第一条执行语句是 happy()函数，这也是一个函数调用。因此，Python 暂停执行 happyB()函数，将控制传递给被调用的函数 happy()。happy()函数体包含了一个简单的 print 语句，该语句执行后函数体结束，程序重新返回调用 happy()函数的位置。图 5.2 给出了 happy()函数调用和返回的执行过程。

```
                              name="Mike"
happyB("Mike") ——→ def happyB(name):
print()                happy() ——→ def happy():
happyB("Lily")         happy()          print("Happy birthday to you!")
                       print("Happy birthday, dear!".format(name))
                       happy()
```

图 5.2　微实例 5.1 中 happy()的被调用和返回过程

程序执行完 happyB()函数体后，返回调用该函数的原始位置，继续执行，如图 5.3 所示。

```
                              name="Mike"
happyB("Mike") ——→ def happyB(name):
print()                happy()
happyB("Lily")         happy()
                       print("Happy birthday, dear!".format(name))
                       happy()
```

图 5.3　微实例 5.1 中 happyB()的被调用和返回过程

拓展： 函数式编程

函数式编程是一种编程范式，常见的编程范式还包括命令式编程和面向对象编程等。函数式编程的主要思想是把程序过程尽量写成一系列函数调用，通过函数进一步提高封装级别。函数式编程通过使用一系列函数能够使代码编写更简洁、更易于理解，是中小规模软件项目中最常用的编程方式。

5.1.3　lambda 函数

表 2.1 介绍了 Python 的 33 个保留字，其中一个是 lambda，该保留字用于定义一种特殊的函数——匿名函数，又称 lambda 函数。匿名函数并非没有名字，而是将函数名作为函数结果返回，语法格式如下：

```
<函数名> = lambda <参数列表>: <表达式>
```

lambda 函数与正常函数一样，等价于下面形式：

```
def <函数名>(<参数列表>):
    return <表达式>
```

简单地说，lambda 函数用于定义简单的、能够在一行内表示的函数，返回一个函数类型，实例如下：

```
>>>f = lambda x, y : x + y
>>>type(f)
<class 'function'>
>>>f(10, 12)
22
```

lambda 函数用于需要函数对象的场景，本书 6.6 节将使用 lambda 函数定义列表的排序原则。

思考与练习

5.1 Python 中定义函数的关键字是（ ）。

A. def B. define C. function D. defunc

5.2 下列不是使用函数的优点的是（ ）。

A. 减少代码重复 B. 使程序更加模块化

C. 使程序便于阅读 D. 为了展现智力优势

5.3 判断题：函数在调用前不需要定义，拿来即用就好。

5.4 下面 Python 程序中定义 f1()时还没有定义 f2()，这种函数调用是否合法？

```
1 def f1():
2     f2()
3 def f2():
4     print("函数 f2()")
5 f1()
```

5.2 函数的参数传递

要点： 函数可以定义可选参数，使用参数的位置或名称传递参数值，根据函数中变量的不同作用域有不同的函数返回值方式。

5.2.1 可选参数和可变数量参数

在定义函数时，如果有些参数存在默认值，即部分参数不一定需要调用程序输入，可以在定义函数时直接为这些参数指定默认值。当函数被调用时，如果没有传入对应的参数值，则使用函数定义时的默认值替代，例如：

```
>>>def dup(str, times = 2):
       print(str*times)
>>>dup("knock~")
knock~knock~
>>>dup("knock~",4)
knock~knock~knock~knock~
```

由于函数调用时需要按顺序输入参数，可选参数必须定义在非可选参数的后面，即 dup()函数中带默认值的可选参数 times 必须定义在 str 参数后面。

在函数定义时，也可以设计可变数量参数，通过在参数前增加星号（*）实现。带有星号的可变参数只能出现在参数列表的最后。调用时，这些参数被当作元组类型传递到函数中，实例如下：

```
>>>def vfunc(a, *b):
       print(type(b))
       for n in b:
           a += n
       return a
>>>vfunc(1,2,3,4,5)
<class 'tuple'>
15
```

vfunc()函数定义了可变参数 b，调用 vfunc()函数时输入的(2, 3, 4, 5)被当作元组传递给 b，与 a 累加后输出。6.1 节将详细介绍元组类型，这里请读者将元组理解为一组元素。

5.2.2　参数的位置和名称传递

函数调用时，实参默认采用按照位置顺序的方式传递给函数，例如 dup ("knock~",4)中第一个实参默认赋值给形参 str，第二个实参赋值给形参 times。这种按照位置传递参数的方法固然很好，但当参数很多时，这种调用参数的方式可读性较差。假设 func()函数有 6 个参数，它的定义如下，其中参数分别表示两组三维坐标值。

```
func(x1, y1, z1, x2, y2, z2):
    return
```

它的一个实际调用如下：

```
result = func(1, 2, 3, 4, 5, 6,)
```

如果仅看实际调用而不看函数定义，很难理解这些输入参数的含义。在规模稍大的程序中，函数定义可能在函数库中，也可能与调用相距很远，带来的可读性较差。

为了解决上述问题，Python 提供了按照形参名称输入实参的方式，此时函数调用如下：

```
result = func(x2=4, y2=5, z2=6, x1=1, y1=2, z1=3)
```

由于调用函数时指定了参数名称，所以参数之间的顺序可以任意调整。

5.2.3　函数的返回值

return 语句用来退出函数并将程序返回到函数被调用的位置继续执行。return 语

句可以同时将 0 个、1 个或多个函数运算后的结果返回给函数被调用处的变量，例如：

```
>>>def func(a, b):
       return a*b
>>>s = func("knock~", 2)
>>>print(s)
knock~knock~
```

函数可以没有 return，此时函数并不返回值，如微实例 5.1 的 happy()函数。函数也可以用 return 返回多个值，多个值以元组类型保存，例如：

```
>>>def func(a, b):
       return b,a
>>>s = func("knock~", 2)
>>>print(s, type(s))
(2, 'knock~') <class 'tuple'>
```

5.2.4 函数对变量的作用

本小节主要讲授函数对程序中变量的作用问题，涉及组合数据类型，建议读者学完第 6 章再阅读本小节。

一个程序中的变量包括两类：全局变量和局部变量。全局变量指在函数之外定义的变量，一般没有缩进，在程序执行全过程有效。局部变量指在函数内部使用的变量，仅在函数内部有效，当函数退出时变量将不存在。例如：

```
>>>n = 1    #n 是全局变量
>>>def func(a, b):
       c = a * b    #c 是局部变量，a 和 b 作为函数参数也是局部变量
       return c
>>>s = func("knock~", 2)
>>>print(c)
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    print(c)
NameError: name 'c' is not defined
```

这个例子说明，当函数执行完退出后，其内部变量将被释放。

如果函数内部使用了全局变量呢？例如：

```
>>>n = 1    #n 是全局变量
>>>def func(a, b):
       n = b    #这个 n 是在函数内存中新生成的局部变量，不是全局变量
       return a*b
```

```
>>>s = func("knock~", 2)
>>>print(s, n)  #测试一下 n 值是否改变
knock~knock~ 1
```

函数 func()内部使用了变量 n，并且将变量参数 b 赋值给变量 n，为何 n 值没有改变？因为函数 func()有自己的内存空间，它将 n=b 语句理解为生成一个局部变量 n，并将参数 b 赋值给它，此时 func()函数没有将 n 当作全局变量。所以，函数退出后，局部变量 n 被释放，全局变量 n 的值没有改变。

如果希望让 func()函数将 n 当作全局变量，需要在变量 n 使用前显式声明该变量为全局变量，代码如下：

```
>>>n = 1    #n 是全局变量
>>>def func(a, b):
       global n
       n = b     #将局部变量 b 赋值给全局变量 n
       return a*b
>>>s = func("knock~", 2)
>>>print(s, n)  #测试一下 n 值是否改变
knock~knock~ 2
```

如果此时的全局变量不是整数 n，而是列表类型 ls，会怎么样呢？理解如下代码：

```
>>>ls = []    #ls 是全局列表变量
>>>def func(a, b):
       ls.append(b)   #将局部变量 b 增加到全局列表变量 ls 中
       return a*b
>>>s = func("knock~", 2)
>>>print(s, ls)  #测试一下 ls 值是否改变
knock~knock~ [2]
```

请读者注意，奇迹产生了，与之前的整数变量 n 不同，全局列表变量在函数 func()调用后竟然发生了改变！

——这是为什么？灵异事件？

——Python 才刚刚开始展现它的魅力。

请读者查看 6.2.1 节内容，列表等组合数据类型由于操作多个数据，所以它们在使用中有创建和引用的分别。当列表变量被方括号（[]，无论是否为空）赋值时，这个列表才被真实创建，否则只是对之前创建列表的一次引用。

上述代码 func()函数的 ls.append(b)语句执行时需要一个真实创建过的列表，此时 func()函数专属的内存空间中没有已经创建过且名称为 ls 的列表，因此，func()函数进一步去寻找全局内存空间，自动关联全局 ls 列表，并修改其内容。当 func()函数退出后，全局 ls 列表中的内容被修改。简单地说，对于列表类型，函数可以直接使用全局列表而不需要采用 global 进行声明。

如果 func()函数内部存在一个真实创建过且名称为 ls 的列表，则 func()函数将操作该列表而不会修改全局变量，例如：

```
>>>ls = []     #ls 是全局列表变量
>>>def func(a, b):
       ls = []      #创建了名称为 ls 的局部列表变量列
       ls.append(b)    #将局部变量 b 增加到全局列表变量 ls 中
       return a*b
>>>s = func("knock~", 3)
>>>print(s, ls)  #测试一下 ls 值是否改变
knock~knock~ []
```

总结一下，Python 函数对变量的作用遵守如下原则。

（1）简单数据类型变量无论是否与全局变量重名，仅在函数内部创建和使用，函数退出后变量被释放，如有全局同名变量，其值不变。

（2）简单数据类型变量在用 global 保留字声明后，作为全局变量使用，函数退出后该变量保留且值被函数改变。

（3）对于组合数据类型的全局变量，如果在函数内部没有被真实创建的同名变量，则函数内部可以直接使用并修改全局变量的值。

（4）如果函数内部真实创建了组合数据类型变量，无论是否有同名全局变量，函数仅对局部变量进行操作，函数退出后局部变量被释放，全局变量值不变。

拓展： 指针和引用

指针是保存内存地址的变量，一般出现在比较底层的程序设计语言中，如 C 语言。引用是某一变量的别名，用这个名字可以对变量进行操作，如 Python 列表类型的引用。两者的主要区别是，指针直接指向内存地址，说明对象已经生成，而引用只是别名，需要真实创建对象才能操作对象。由于列表类型在 Python 中十分常用，要格外注意该类型真实创建和引用的区别。

程序练习 5-1：半小时学 Python 函数使用



思考与练习

5.5　如何定义带有可选参数的函数？

5.6　如何定义带有可变数量参数的函数？

5.7　假如 return 语句同时返回 3 个值，返回值是什么数据类型？

5.8　参数的位置传递和名称传递各有什么优缺点？

5.9　在函数中操作全局列表类型变量时需要注意什么问题？

阶段测试 5-1：Python 函数概念小测验



5.3　模块 3：datetime 库的使用

要点： Python 时间处理的标准函数库 datetime 提供了一批显示日期和时间的格式化方法。

5.3.1 datetime 库概述

图片资料 5-1：Python 快速参考之 random、jieba、datetime 库



以不同格式显示日期和时间是程序中最常用到的功能。Python 提供了一个处理时间的标准函数库 datetime，它提供了一系列由简单到复杂的时间处理方法。datetime 库可以从系统中获得时间，并以用户选择的格式输出。

datetime 库以格林威治时间为基础，每天由 3 600×24 秒精准定义。该库包括两个常量：datetime.MINYEAR 与 datetime.MAXYEAR，分别表示 datetime 所能表示的最小、最大年份，值分别为 1 与 9 999。

datetime 库以类的方式提供多种日期和时间表达方式。

（1）datetime.date：日期表示类，可以表示年、月、日等。

（2）datetime.time：时间表示类，可以表示小时、分钟、秒、毫秒等。

（3）datetime.datetime：日期和时间表示的类，功能覆盖 date 和 time 类。

（4）datetime.timedelta：与时间间隔有关的类。

（5）datetime.tzinfo：与时区有关的信息表示类。

由于 datetime.datetime 类表达形式最为丰富，这里主要介绍这个类的使用。使用 datetime 类需要用 import 保留字，引用 datetime 类的方式如下：

```
1  from datetime import datetime
```

拓展： 1970 年 1 月 1 日

当代计算机系统都有一个计时功能，能够输出从格林威治标准时间 1970 年 1 月 1 日 00:00:00 开始到当下的时间计数，精确到秒，这是 UNIX 操作系统早期的设计习惯，后沿用到所有计算机系统中。

现在的计算机硬件和系统都是 64 位，如果用 64 位存储这个时间计数则最大可以表示距离 1970 年 1 月 1 日开始的 2^{64} 秒，1 年 365 天的总秒数约为 1.9×2^{24}，因此，64 位计算机系统可以将时间表示到约公元 2^{39} 年，相信我们的 N 代子孙，哪怕到地球毁灭那天都不用担心时间不准确了。

——为什么选择从 1970 年 1 月 1 日开始？

——无论选择从哪天开始，都会有同样的问题，不是吗？

5.3.2 datetime 库解析

datetime 类（datetime.datetime 类，以下简称为 datetime 类）的使用方式是首先创建一个 datetime 对象，然后通过对象的方法和属性显示时间。创建 datetime 对象有 3 种方法：datetime.now()、datetime.utcnow()和 datetime.datetime()。

1．使用 datetime.now()获得当前日期和时间对象，使用方法如下：

```
datetime.now()
```

作用：返回一个 datetime 类型，表示当前的日期和时间，精确到微秒。

参数：无

调用该函数，执行结果如下：

```
>>> from datetime import datetime
>>> today = datetime.now()
>>> today
datetime.datetime(2016, 9, 20, 10, 29, 43, 928549)
```

2．使用 datetime.utcnow()获得当前日期和时间对应的 UTC（世界标准时间）时间对象，使用方法如下：

datetime.utcnow()

作用：返回一个 datetime 类型，表示当前日期和时间的 UTC 表示，精确到微秒。

参数：无

调用该函数，执行结果如下：

```
>>> today = datetime.utcnow()
>>> today
datetime.datetime(2016, 9, 20, 2, 35, 1, 427954)
```

3．datetime.now()和 datetime.utcnow()都返回一个 datetime 类型的对象，也可以直接使用 datetime()构造一个日期和时间对象，使用方法如下：

datetime(year, month, day, hour=0, minute=0,second=0, microsecond=0)

作用：返回一个 datetime 类型，表示指定的日期和时间，可以精确到微秒。

参数如下。

year：指定的年份，MINYEAR <= year <= MAXYEAR

month：指定的月份，1 <= month <= 12

day：指定的日期，1 <= day <= 月份所对应的日期上限

hour：指定的小时，0 <= hour < 24

minute：指定的分钟数，0 <= minute < 60

second：指定的秒数，0 <= second < 60

microsecond：指定的微秒数，0 <= microsecond <1000000

其中，hour、minute、second、microsecond 参数可以全部或部分省略。

调用 datetime()函数直接创建一个 datetime 对象，表示 2016 年 9 月 16 日 22:33，32 秒 7 微秒，执行结果如下：

```
>>> someday = datetime(2016,9,16,22,33,32,7)
>>> someday
datetime.datetime(2016, 9, 16, 22, 33, 32, 7)
```

到此，程序已经有了一个 datetime 对象，进一步可以利用这个对象的属性显示时间，为了区别 datetime 库名，采用上例中的 someday 代替生成的 datetime 对象，常用属性如表 5.1 所示。

表 5.1　datetime 类的常用属性（共 9 个）

属　　性	描　　述
someday.min	固定返回 datetime 的最小时间对象，datetime(1,1,1,0,0)
someday.max	固定返回 datetime 的最大时间对象，datetime(9999,12,31,23,59,59,999999)
someday.year	返回 someday 包含的年份
someday.month	返回 someday 包含的月份
someday.day	返回 someday 包含的日期
someday.hour	返回 someday 包含的小时
someday.minute	返回 someday 包含的分钟
someday.second	返回 someday 包含的秒钟
someday.microsecond	返回 someday 包含的微秒值

datetime 对象有 3 个常用的时间格式化方法，如表 5.2 所示。

表 5.2　datetime 类常用的时间格式化方法（共 3 个）

属　　性	描　　述
someday.isoformat()	采用 ISO 8601 标准显示时间
someday.isoweekday()	根据日期计算星期后返回 1~7,对应星期一到星期日
someday.strftime(format)	根据格式化字符串 format 进行格式显示的方法

isoformat()和 isoweekday()方法的使用如下：

```
>>> someday = datetime(2016,9,16,22,33,32,7)
>>> someday.isoformat()
'2016-09-16T22:33:32.000007'
>>> someday.isoweekday()
5
```

strftime()方法是时间格式化最有效的方法，几乎可以以任何通用格式输出时间。例如下例所示，用该方法输出特定格式时间。表 5.3 给出了 strftime()方法的格式化控制符。

```
>>> someday.strftime("%Y-%m-%d %H:%M:%S")
'2016-09-16 22:33:32'
```

表 5.3　strftime()方法的格式化控制符

格式化字符串	日期/时间	值范围和实例
%Y	年份	0001~9999，例如，1900
%m	月份	01~12，例如，10
%B	月名	January~December，例如，April
%b	月名缩写	Jan~Dec，例如，Apr
%d	日期	01 ~ 31，例如，25
%A	星期	Monday~Sunday，例如，Wednesday
%a	星期缩写	Mon~Sun，例如，Wed
%H	小时（24 h 制）	00 ~ 23，例如，12
%I	小时（12 h 制）	01 ~ 12，例如，7
%p	上/下午	AM, PM，例如，PM
%M	分钟	00 ~ 59，例如，26
%S	秒	00 ~ 59，例如：26

strftime()格式化字符串的数字左侧会自动补零，上述格式也可以与 print()的格式化函数一起使用，例如：

```
>>>from datetime import datetime
>>>now = datetime.now()
>>>now.strftime("%Y-%m-%d")
'2016-09-20'
>>>now.strftime("%A, %d. %B %Y %I:%M%p")
'Tuesday, 20. September 2016 01:53PM'
>>>print("今天是{0:%Y}年{0:%m}月{0:%d}日".format(now))
今天是 2016 年 09 月 20 日
```

datetime 库主要用于对时间的表示，从格式化角度掌握 strftime()函数已经能够处理很多情况了。建议读者在遇到需要处理时间的问题时采用 datetime 库，简化格式输出和时间的维护。





思考与练习

5.10 请利用 datetime 库将当前系统时间转换为字符串。

5.11 请利用 datetime 库输出 5 种不同的日期格式。

5.12 思考如何利用 datetime 库对一个程序的运行计时。

5.4 实例 7：七段数码管绘制

> **要点：** 这是一个绘制七段数码管的实例，用于理解函数及其封装的价值。

数码管是一种价格便宜、使用简单的发光电子器件，广泛应用在价格较低的电子类产品中，其中，七段数码管最为常用。七段数码管（Seven-segment Indicator）由 7 段数码管拼接而成，每段有亮或不亮两种情况，改进型的七段数码管还包括一个小数点位置，如图 5.4 所示。

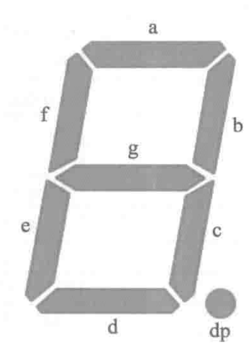


图 5.4 七段数码管的结构图

七段数码管能形成 2^7=128 种不同状态，其中部分状态能够显示易于人们理解的数字或字母含义，因此被广泛使用。图 5.5 给出了十六进制中 16 个字符的七段数码管表示。

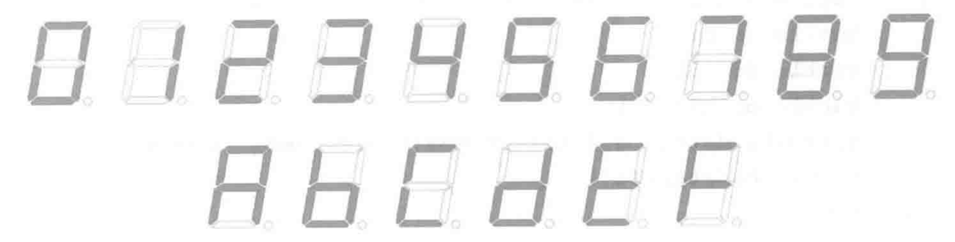


图 5.5　十六进制中 16 个字符的七段数码管表示

本节将延续实例 2 和 2.4 节内容，通过 turtle 库函数绘制七段数码管形式的日期信息。该问题的 IPO 描述如下。

输入：当前日期的数字形式

处理：根据每个数字绘制七段数码管表示

输出：绘制当前日期的七段数码管表示

每个 0 到 9 的数字都有相同的七段数码管样式，因此，可以通过设计函数复用数字的绘制过程。进一步，每个七段数码管包括 7 个数码管样式，除了数码管位置不同外，绘制风格一致，也可以通过函数复用单个数码段的绘制过程。这里，先给出程序的全部代码，实例代码 7.1 如下。

源代码 5-2：基本的七段数码管绘制



实例代码 7.1　　　　e7.1DrawSevenSegDisplay.py

```
 1  #e7.1DrawSevenSegDisplay.py
 2  import turtle, datetime
 3  def drawLine(draw):   #绘制单段数码管
 4      turtle.pendown() if draw else turtle.penup()
 5      turtle.fd(40)
 6      turtle.right(90)
 7  def drawDigit(d): #根据数字绘制七段数码管
 8      drawLine(True) if d in [2,3,4,5,6,8,9] else drawLine(False)
 9      drawLine(True) if d in [0,1,3,4,5,6,7,8,9] else drawLine(False)
10      drawLine(True) if d in [0,2,3,5,6,8,9] else drawLine(False)
11      drawLine(True) if d in [0,2,6,8] else drawLine(False)
12      turtle.left(90)
13      drawLine(True) if d in [0,4,5,6,8,9] else drawLine(False)
14      drawLine(True) if d in [0,2,3,5,6,7,8,9] else drawLine(False)
15      drawLine(True) if d in [0,1,2,3,4,7,8,9] else drawLine(False)
16      turtle.left(180)
17      turtle.penup()
18      turtle.fd(20)
19  def drawDate(date):  #获得要输出的数字
```

```
20      for i in date:
21          drawDigit(eval(i))  #注意：通过 eval()函数将数字变为整数
22  def main():
23      turtle.setup(800, 350, 200, 200)
24      turtle.penup()
25      turtle.fd(-300)
26      turtle.pensize(5)
27      drawDate(datetime.datetime.now().strftime('%Y%m%d'))
28      turtle.hideturtle()
29  main()
```

实例代码 7.1 定义了 drawDigit()函数，该函数根据输入的数字 d 绘制七段数码管，结合七段数码管结构，每个数码管的绘制采用图 5.6 所示的顺序。

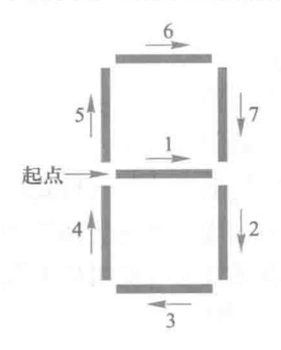


图 5.6 七段数码管的绘制顺序

绘制起点在数码管中部左侧，无论每段数码管是否被绘制出来，turtle 画笔都按顺序“画完”7 个数码管。对于给定数字 d，哪个数码段被绘制出来采用 if-else 语句判断。

```
8   drawLine(True) if d in [2,3,4,5,6,8,9] else drawLine(False)
```

以第 8 行为例，代码采用了单行 if-else 语句，这种语句常用于 if 和 else 分别只有一行语句的情形。第 8 行代码采用普通 if-else 语句表达如下，可见，单行语句的实现方式能够使表达更加紧凑。

```
1   if d in [2,3,4,5,6,8,9]:
2       drawLine(True)
3   else:
4       drawLine(False)
```

第 8 行代码根据输入数字判断是否要绘制七段数码管最中间的横线，当需要绘

制时，调用绘制函数 drawLine()，参数赋值 True；当不需要绘制时，参数赋值 False。根据 0～9 数字结构，对于 2、3、4、5、6、8、9 这些数字需要绘制，否则不需要绘制。为了使输出样式固定，简化设计，当不需要绘制时，turtle 画笔需要抬起。

drawLine()函数根据输出参数的值（True 或 False）决定是否抬起画笔。

为了使代码模块化更好，实例代码 7.1 定义了 drawDate()函数和 main()函数。其中，drawDate()函数将更长数字分解为单个数字，进一步调用 drawDigit()分别绘制每个数字。main()函数将启动窗体大小、设置画笔宽度、设置系统时间等功能封装在一起，但 main()函数并不体现单一功能，这种封装仅从提高代码可读性角度考虑。

请读者逐行理解实例代码 7.1 的其余部分，该程序运行后的效果如图 5.7 所示。



图 5.7　实例代码 7.1 的运行效果

实例代码 7.1 给出了最基本的七段数码管绘制程序，可以看出，使用函数能大量复用代码，避免相同功能重复编写。此外，函数的好处还体现在对代码的修改方面。能否绘制更有趣的七段数码管呢？实例代码 7.2 给出了图 5.8 的绘制风格，请读者比较实例代码 7.2 和实例代码 7.1，进一步体会函数为编程带来的便利。



图 5.8　实例代码 7.2 的运行效果

源代码 5-3：风格的七段数码管绘制

实例代码 7.2　　e7.2DrawSevenSegDisplay.py

```
1  #e7.2DrawSevenSegDisplay.py
2  import turtle, datetime
3  def drawGap(): #绘制数码管间隔
4      turtle.penup()
5      turtle.fd(5)
6  def drawLine(draw):   #绘制单段数码管
7      drawGap()
8      turtle.pendown() if draw else turtle.penup()
9      turtle.fd(40)
10     drawGap()
11     turtle.right(90)
12 def drawDigit(d): #根据数字绘制七段数码管
13     drawLine(True) if d in [2,3,4,5,6,8,9] else drawLine(False)
14     drawLine(True) if d in [0,1,3,4,5,6,7,8,9] else drawLine(False)
15     drawLine(True) if d in [0,2,3,5,6,8,9] else drawLine(False)
16     drawLine(True) if d in [0,2,6,8] else drawLine(False)
17     turtle.left(90)
18     drawLine(True) if d in [0,4,5,6,8,9] else drawLine(False)
19     drawLine(True) if d in [0,2,3,5,6,7,8,9] else drawLine(False)
20     drawLine(True) if d in [0,1,2,3,4,7,8,9] else drawLine(False)
21     turtle.left(180)
22     turtle.penup()
23     turtle.fd(20)
24 def drawDate(date):
25     turtle.pencolor("red")
26     for i in date:
27         if i == '-':
28             turtle.write('年',font=("Arial", 18, "normal"))
29             turtle.pencolor("green")
30             turtle.fd(40)
31         elif i == '=':
32             turtle.write('月',font=("Arial", 18, "normal"))
33             turtle.pencolor("blue")
34             turtle.fd(40)
35         elif i == '+':
36             turtle.write('日',font=("Arial", 18, "normal"))
37         else:
38             drawDigit(eval(i))
39 def main():
40     turtle.setup(800, 350, 200, 200)
41     turtle.penup()
42     turtle.fd(-350)
43     turtle.pensize(5)
44     drawDate(datetime.datetime.now().strftime('%Y-%m=%d+'))
45     turtle.hideturtle()
46 main()
```

拓展： 计算机的硬件时钟

计算机断电再次打开后其系统时间往往是准确的，既然已经断电，计算机如何能准确计时呢？原来，计算机主板上有一颗纽扣电池，它负责给硬件时钟供电。这颗纽扣电池往往能用几年，所以，硬件时钟产生的时间都是准确且连续的。计算机的硬件时钟由硬件计时电路组成。当需要使用时间时，操作系统会从硬件时钟中读出时间放入内核给应用软件使用。

思考与练习

5.13　请说明单行 if-else 语句有哪些作用？

5.14　请查阅资料回答实例代码 7.1 中第 28 行代码的作用。

5.15　为什么实例代码 7.2 中第 44 行使用'%Y-%m=%d+'作为格式化样式？

5.5　代码复用和模块化设计

要点： 函数是程序的一种抽象，它通过封装实现代码复用。可以利用函数对程序进行模块化设计。

程序由一系列代码组成，如果代码是顺序但无组织的，不仅不利于阅读和理解，也很难进行升级和维护。因此，需要对代码进行抽象，形成易于理解的结构。当代编程语言从代码层面采用函数和对象两种抽象方式，分别对应面向过程和面向对象编程思想。

函数是程序的一种基本抽象方式，它将一系列代码组织起来通过命名供其他程序使用。函数封装的直接好处是代码复用，任何其他代码只要输入参数即可调用函数，从而避免相同功能代码在被调用处重复编写。代码复用产生了另一个好处，当更新函数功能时，所有被调用处的功能都被更新。

面向过程是一种以过程描述为主要方法的编程方式，该方法要求程序员列出解决问题所需要的步骤，然后用函数将这些步骤一步一步实现，使用时依次建立并调用函数或编写语句即可。面向过程编程是一种基本且自然的程序设计方法，函数通过将步骤或子功能封装实现代码复用并简化程序设计难度。

对象是程序的一种高级抽象方式，它将程序代码组织为更高级别的类。对象包括表征对象特征的属性和代表对象操作的方法。例如，汽车是一个对象，其颜色、轮胎数量、车型是属性，代表汽车的静态值；前进、后退、转弯等是方法，代表汽车的动作和行为。在程序设计中，如果<a>代表对象，获取其属性<b>采用<a>.<b>，调用其方法<c>采用<a>.<c>()。对象的方法具有程序功能性，因此采用函数形式封装。简单地，对象是程序拟解决计算问题的一个高级别抽象，它包括一组静态值（属性）和一组函数（方法）。从代码行数角度来看，对象和函数都使用了一个容易理解的抽象逻辑，但对象可以凝聚更多代码。因此，面向对象编程更适合代码规模较大，

交互逻辑复杂的程序。

面向过程和面向对象只是编程方式不同、抽象级别不同，所有面向对象编程能实现的功能采用面向过程同样能完成，两者在解决问题上不存在优劣之分，很多专业程序员仅采用面向过程方式编程。具体采用哪种方法取决于具体开发环境和要求，一般在编写较大规模程序时采用面向对象方法，如 Windows 操作系统，或需要 10 人或更多人协同开发的程序，或带有窗口交互类的程序。Python 语言同时支持面向过程和面向对象两种编程方式，本书仅介绍但不讲解面向对象编程。

当程序的长度在百行以上，如果不划分模块，程序的可读性非常糟糕。解决这一问题的最好方法是将一个程序分隔成短小的程序段，每一段程序完成一个小的功能。无论面向过程还是面向对象编程，对程序合理划分功能模块并基于模块设计程序是一种常用方法，被称为“模块化设计”。

模块化设计指通过函数或对象的封装功能将程序划分成主程序、子程序和子程序间关系的表达。模块化设计是使用函数和对象设计程序的思考方法，以功能块为基本单位，一般有以下两个基本要求。

（1）紧耦合：尽可能合理划分功能块，功能块内部耦合紧密。

（2）松耦合：模块间关系尽可能简单，功能块之间耦合度低。

使用函数只是模块化设计的必要非充分条件，根据计算需求合理划分函数十分重要。一般来说，完成特定功能或被经常复用的一组语句应该采用函数来封装，并尽可能减少函数间参数和返回值的数量。

拓展： 紧耦合和松耦合

耦合性指程序结构中各模块之间相互关联的程度，它取决于各模块间接口的复杂程度和调用方式。耦合性是影响软件复杂程度和设计质量的一个重要因素。紧耦合指模块或系统间关系紧密，存在较多或复杂的相互调用。紧耦合的缺点在于更新一个模块可能导致其他模块变化，复用较困难。松耦合一般基于消息或协议实现，系统间交互简单。

对比实例代码 2.1 和实例代码 2.3，尽管后者通过函数封装增多了代码行数，但采用 drawSnake()函数封装的功能理解起来更加容易。观察实例代码 7.1，采用函数封装后，理解程序的第一层次不再是直接阅读语句，而是阅读函数及其框架，有需要时再进一步理解函数内部语句。为了增强代码可读性，建议读者采用实例代码 7.1 中的方式，将程序初始化或函数间过渡语句封装为 main()函数，使得全部代码都由函数组成，最后通过调用 main()函数执行程序。

尽管函数和模块化设计使整个程序看上去篇幅更长，但所增加的封装代码十分有益，它们为程序带来了模块化的层次结构，使程序具有更好的可读性。

思考与练习

5.16 判断题：使用函数封装的程序一定符合模块化设计原则。

5.17 判断题：使用函数一定能够简化程序理解，没什么弊端。

5.18 思考松耦合和紧耦合在实例 7 中的体现。

5.6　函数的递归

> **要点：** 函数定义中调用函数自身的方式形成递归。

5.6.1　递归的定义

函数作为一种代码封装，可以被其他程序调用，当然，也可以被函数内部代码调用。这种函数定义中调用函数自身的方式称为递归。就像一个人站在装满镜子的房间中，看到的影像就是递归的结果。递归在数学和计算机应用上非常强大，能够非常简洁地解决重要问题。

数学上有个经典的递归例子叫阶乘，阶乘通常定义如下：

$$n! = n(n-1)(n-2)\cdots(1)$$

为了实现这个程序，可以通过一个简单的循环累积去计算阶乘。观察 5!的计算，如果去掉了 5，那么就剩下计算 4!，推广来看，$n!=n(n-1)!$。实际上，这个关系给出了另一种表达阶乘的方式：

$$n! = \begin{cases} 1 & n = 0 \\ n(n-1)! & \text{otherwise} \end{cases}$$

这个定义说明 0 的阶乘按定义是 1，其他数字的阶乘定义为这个数字乘以比这个数字小 1 数的阶乘。递归不是循环，因为每次递归都会计算比它更小数的阶乘，直到 0!。0!是已知的值，被称为递归的基例。当递归到底了，就需要一个能直接算出值的表达式。

阶乘的例子揭示了递归的两个关键特征。

（1）存在一个或多个基例，基例不需要再次递归，它是确定的表达式。

（2）所有递归链要以一个或多个基例结尾。

> **拓展：** 数学归纳法
>
> 数学归纳法和递归都利用了递推原理，本质是相同的。在证明一个与自然数相关的命题 P(n)时，数学归纳法采用如下步骤。
>
> （1）证明当 n 取第一个值 n_0 时命题成立。
>
> （2）假设当 n_k（$k \geqslant 0$，k 为自然数）时命题成立，证明当 $n=n_{k+1}$ 时命题也成立。
>
> 综合（1）和（2），对一切自然数 $n(n \geqslant n_0)$，命题 P(n) 都成立。

5.6.2　递归的使用方法

以阶乘计算为例，可以把阶乘写成一个单独的函数，则该函数如微实例 5.2 中第 1 到第 5 行所示。

【微实例 5.2】阶乘的计算。

源代码 5-4：
阶乘的计算



根据用户输入的整数 *n*，计算并输出 *n* 的阶乘值。

微实例 5.2　　　　m5.2CalFactorial.py

```
1  def fact(n):
2      if n == 0:
3          return 1
4      else:
5          return n * fact(n-1)
6  num = eval(input("请输入一个整数: "))
7  print(fact(abs(int(num))))
```

fact()函数在其定义内部引用了自身，形成了递归过程（如第 5 行）。无限制的递归将耗尽计算资源，因此，需要设计基例使得递归逐层返回。fact()函数通过 if 语句给出了 n 为 0 时的基例，当 n==0，fact()函数不再递归，返回数值 1，如果 n!=0，则通过递归返回 n 与 n−1 阶乘的乘积。

由于负数和小数通过减 1 无法到达递归的基例（n==0），代码第 7 行通过 abs()和 int()函数将用户输入转变成非负整数，该程序输出效果如下：

```
>>>
请输入一个整数: 10
3628800
>>>
请输入一个整数: 12.345
479001600
```

递归遵循函数的语义，每次调用都会引起新函数的开始，表示它有本地变量值的副本，包括函数的参数。图 5.9 给出了计算 5!的递归调用过程，每次函数调用时，函数参数的副本会临时存储，递归中各函数再运算自己的参数，相互没有影响。当基例结束运算并返回值时，各函数逐层结束运算，向调用者返回计算结果。

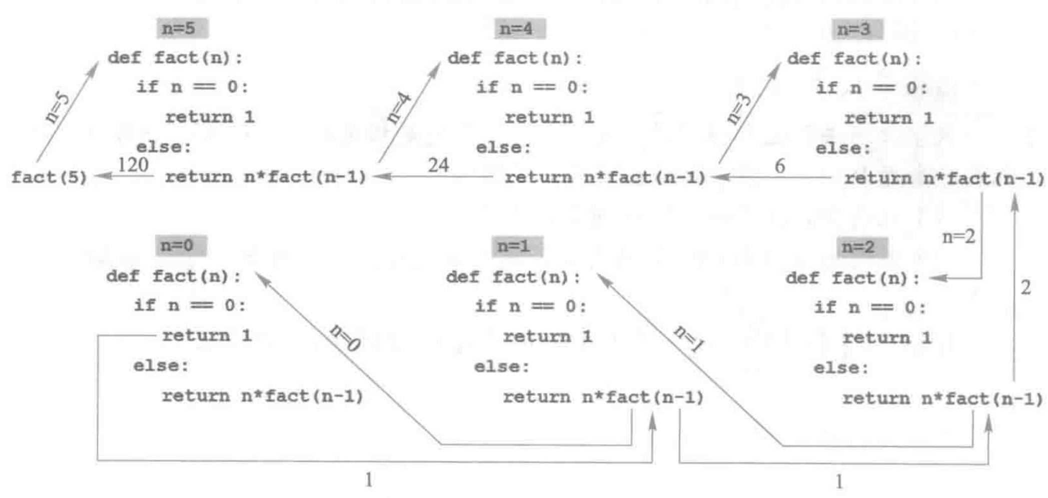


图 5.9　阶乘 5!的递归调用过程

使用递归一定要注意基例的构建，否则递归无法返回将会报错。

【微实例 5.3】字符串反转。

对于用户输入的字符串 s，输出反转后的字符串。

解决这个问题的基本思想是把字符串看作一个递归对象。长字符串由较短字符串组成，每个小字符串也是一个对象。假如把一个字符串看成仅由两部分组成：首字符和剩余字符串。如果将剩余字符串与首字符交换，就完成了反转整个字符串，代码如下：

```
1  def reverse(s):
2      return reverse(s[1:]) + s[0]
```

观察这个函数的工作过程。s[0]是首字符，s[1:]是剩余字符串，将它们反向连接，可以得到反转字符串。执行这个程序，结果如下：

```
>>>def reverse(s):
       return reverse(s[1:]) + s[0]
>>> reverse("ABC")
...
RecursionError: maximum recursion depth exceeded
```

这个错误表明系统无法执行 reverse()函数创建的递归，这是因为 reverse()函数没有基例，递归层数超过了系统允许的最大递归深度。默认情况下，当递归调用到 1 000 层，Python 解释器将终止程序。递归深度是为了防止无限递归错误而设计的，当用户编写的正确递归程序需要超过 1 000 层时，可以通过如下代码设定。

```
>>>import sys
>>>sys.setrecursionlimit(2000)  #2000 是新的递归层数
```

reverse()超过递归深度是因为没有设计基例。字符串反转中的递归调用总是使用比之前更短的字符串，因此，可以把基例设计为字符串的最短形式，即空字符串。微实例 5.3 的完整代码如下：

源代码 5–5：字符串反转



微实例 5.3　　m5.3ReverseString.py

```
1  def reverse(s):
2      if s == "":
3          return s
4      else:
5          return reverse(s[1:]) + s[0]
6  str = input("请输入一个字符串：")
7  print(reverse(str))
```

微实例 5.3 的结果如下：

```
>>>
请输入一个字符串: 唐诗宋词
词宋诗唐
```

阶段测试 5-2：Python 函数递归小测验



思考与练习

5.19 下列不是递归程序特点的是（ ）。

A. 书写简单 B. 一定要有基例

C. 执行效率高 D. 思路简单，代码不一定容易理解

5.20 有哪些思路可以构建递归的基例？

5.21 递归和循环有什么区别？

5.7 实例 8：科赫曲线绘制

要点： 这是一个采用递归方法绘制科赫曲线的实例，分形几何采用类似递归的核心思想。

自然界有很多图形很规则，符合一定的数学规律，例如，蜜蜂蜂窝是天然的等边六角形等。科赫(Koch)曲线在众多经典数学曲线中非常著名，由瑞典数学家冯 •科赫(H·V·Koch)于 1904 年提出，由于其形状类似雪花，也被称为雪花曲线。

科赫曲线的基本概念和绘制方法如下。

正整数 n 代表科赫曲线的阶数，表示生成科赫曲线过程的操作次数。科赫曲线初始化阶数为 0，表示一个长度为 L 的直线。对于直线 L，将其等分为 3 段，中间一段用边长为 $L/3$ 的等边三角形的两个边替代，得到 1 阶科赫曲线，它包含 4 条线段。进一步对每条线段重复同样的操作后得到 2 阶科赫曲线。继续重复同样的操作 n 次可以得到 n 阶科赫曲线，如图 5.10 所示。

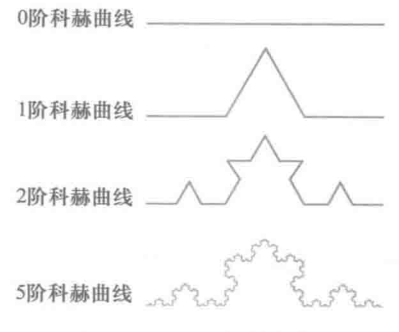


图 5.10 n 阶科赫曲线

科赫曲线属于分形几何分支，它的绘制过程体现了递归思想，绘制过程代码如下：

实例代码 8.1　　　　e8.1DrawKoch.py




```
1   #e8.1DrawKoch.py
2   import turtle
3   def koch(size, n):
4       if n == 0:
5           turtle.fd(size)
6       else:
7           for angle in [0, 60, -120, 60]:
8              turtle.left(angle)
9              koch(size/3, n-1)
10  def main():
11      turtle.setup(800,400)
12      turtle.speed(0)            # 控制绘制速度
13      turtle.penup()
14      turtle.goto(-300, -50)
15      turtle.pendown()
16      turtle.pensize(2)
17      koch(600,3)      # 0 阶科赫曲线长度，阶数
18      turtle.hideturtle()
19  main()
```

n 阶科赫曲线的绘制相当于在画笔前进方向的 0°、60°、-120° 和 60° 分别绘制 *n*-1 阶曲线。实例代码 8.1 中 main()函数设置了一些初始参数，如果希望控制绘制科赫曲线的速度，可以采用 turtle.speed()函数增加或减少速度。

科赫曲线从一条直线绘制开始，如果从倒置的三角形开始将更有趣。替换实例代码 8.1 中的 main()函数，代码如下。在给定初始图形后，通过科赫曲线可以生成很多漂亮的图形，如图 5.11 所示，请读者探索和尝试。

实例代码 8.2　　　　e8.2DrawKoch.py




```
1   #e8.2DrawKoch.py
2   import turtle
3   def koch(size, n):
4       if n == 0:
5           turtle.fd(size)
6       else:
7           for angle in [0, 60, -120, 60]:
8              turtle.left(angle)
9              koch(size/3, n-1)
10  def main():
11      turtle.setup(600,600)
12      turtle.speed(0)
```

```
13        turtle.penup()
14        turtle.goto(-200,100)
15        turtle.pendown()
16        turtle.pensize(2)
17        level=5
18        koch(400,level)
19        turtle.right(120)
20        koch(400,level)
21        turtle.right(120)
22        koch(400,level)
23        turtle.hideturtle()
24    main()
```

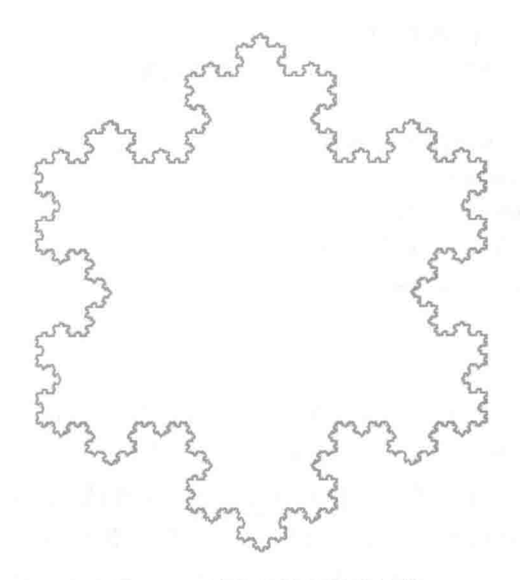


图 5.11 科赫曲线的雪花效果

拓展： 分形几何

分形几何学是数学的一个分支，以不规则几何形态为研究对象。分形以自相似结构为基础，通过无限递归方式展示复杂表面下的内在数学秩序。分形几何不仅展示了数学之美，也揭示了世界的本质，使人们重新审视这个世界：世界是非线性的，分形无处不在。

思考与练习

5.22 如何改变 turtle 绘制过程的速度？

5.23 修改实例代码 8.1，使科赫曲线反向绘制，从直线开始，中间部分向下方绘制。

5.24 修改实例代码 8.1，修改科赫曲线的绘制颜色。

5.8　Python 内置函数

> **要点：** Python 提供 68 个内置函数，需要读者掌握其中的 36 个。

Python 解释器提供了 68 个内置函数，这些函数不需要引用库直接使用，如表 5.4 所示。其中，前 36 个已经或将在本书中出现，需要读者掌握。

表 5.4　Python 的内置函数列表（共 68 个）

abs()	id()	round()	compile()	locals()
all()	input()	set()	dir()	map()
any()	int()	sorted()	exec()	memoryview()
asci()	len()	str()	enumerate()	next()
bin()	list()	tuple()	filter()	object()
bool()	max()	type()	format()	property()
chr()	min()	zip()	frozenset()	repr()
complex()	oct()		getattr()	setattr()
dict()	open()		globals()	slice()
divmod()	ord()	bytes()	hasattr()	staticmethod()
eval()	pow()	delattr()	help()	sum()
float()	print()	bytearray()	isinstance()	super()
hash()	range()	callable()	issubclass()	vars()
hex()	reversed()	classmethod()	iter()	__import__()

部分函数说明如下，其他尚未遇到的函数将在后续章节中介绍并使用。

all()函数一般针对组合数据类型，如果其中每个元素都是 True，则返回 True，否则返回 False。需要注意的是，整数 0、空字符串""、空列表[]等都被当作 False。

any()函数与 all()函数相反，只要组合数据类型中任何一个是 True，则返回 True，全部元素都是 False 时返回 False。

hash()函数对于能够计算哈希的类型返回哈希值。

id()函数对每一个数据返回唯一编号，数据不同编号不同，可以通过比较两个变量编号是否相同判断数据是否一致。Python 将数据存储在内存中的地址作为其唯一编号。

reversed()函数返回输入组合数据类型的逆序形式。

sorted()函数对一个序列进行排序，默认从小到大排序。

type()函数返回每个数据对应的类型。

上述函数的一些实例如下：

```
>>>ls = [1, 2, 5, 0]
>>>all(ls)
```

```
False
>>>any(ls)
True
>>>hash("中国, 你好")
1050901099
>>>id(ls)
45047400
>>>id("中国, 你好")
47174048
>>>list(reversed(ls))
[0, 5, 2, 1]
>>> sorted(ls)   #不改变 ls 的值
[0, 1, 2, 5]
>>>ls
[1, 2, 5, 0]
>>>sorted(ls, reverse=True)
[5, 2, 1, 0]
>>>type(ls)
<class 'list'>
>>>type(reverse(ls))
<class 'list_reverseiterator'>
```

拓展： Python 使用手册

Python 语言具有丰富的内置数据类型、函数和标准库，更多资料请查阅 Python 使用手册。Python 使用手册由一组文档构成，包括 Tutorial、Library Reference、Laguage Reference、Python Setup and Usage、Python HOWTOs、Installing Python Modules、Distributing Python Modules、Extending and Embedding、Python/C API 以及 FAQs 等，读者可以访问 https://docs.python.org/3 阅读并下载。

思考与练习

5.25 请使用 type()函数分别对整数、浮点数、字符串进行类型判断。

5.26 请使用 len()函数对整数、浮点数、字符串进行类型长度计算，解释看到的结果。

5.27 请使用 hex()函数分别计算整数 1 024、12 800，65 536 的十六进制。

本 章 小 结

本章主要介绍了函数及代码复用问题，包括函数的定义、lambda 函数使用、函数递归以及参数的位置和名称传递等内容。本章还介绍了 datetime 时间日期库的使

用，并绘制了七段数码管时钟，讲解了如何使用函数递归绘制复杂精美的科赫曲线。

程序练习 5-3：
章节程序练习题



程序练习题

5.1　程序练习题 3.5 输出了一个简单的田字格，用函数简化其代码，输出如图 5.12 所示的更大田字格。

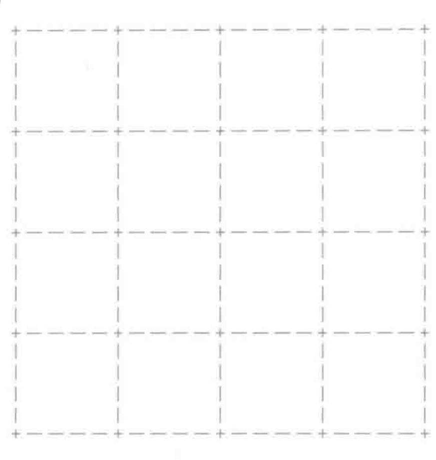


图 5.12　更大的田字格

5.2　实现 isOdd()函数，参数为整数，如果整数为奇数，返回 True，否则返回 False。

5.3　实现 isNum()函数，参数为一个字符串，如果这个字符串属于整数、浮点数或复数的表示，则返回 True，否则返回 False。

5.4　实现 multi()函数，参数个数不限，返回所有参数的乘积。

5.5　实现 isPrime()函数，参数为整数，要有异常处理。如果整数是质数，返回 True，否则返回 False。

5.6　使用 datetime 库，对自己的生日输出不少于 10 种日期格式。

5.7　汉诺塔是学习计算机递归算法的经典入门案例，该案例来源于真实故事。在世界某个地方有一个很虔诚的宗教组织，其中僧侣维护着一项神圣任务：保持宇宙的时间。在时间的最开始，僧侣在平台上竖立了 3 个垂直杆，在最左侧杆上有 64 个不同半径的金色同心圆盘，直径较大的圆盘堆放在下方，形成了金字塔样子的整体外观。僧侣们的任务是将所有圆盘从最左侧杆子移动到最右侧杆子上，这个宗教认为当僧侣们完成任务的时候，万事万物将会化为乌有，宇宙将结束。为了保持神圣的顺序，僧侣们移动圆盘需要遵从特定的规则：一次只能移动一个盘子、盘子只能在 3 个标杆之间移动、更大的盘子不能放在更小的盘子上面。图 5.13 给出了汉诺塔问题的示例图，其中，3 个标杆分别用 A、B 和 C 表示。

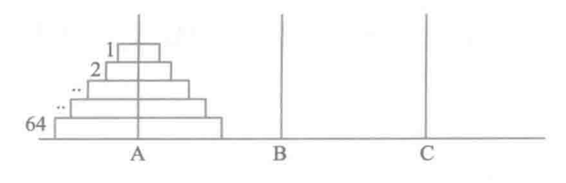


图 5.13 汉诺塔问题的示例图

汉诺塔是一个数学难题，其问题描述为如何将所有圆盘从 A 移动到 C。请用 Python 编写一个汉诺塔的移动函数，采用递归方法解决这个难题，要求输入汉诺塔的层数，输出整个移动流程。

电子教案 6-1：
组合数据类型



第 6 章　组合数据类型

计算机根据指令执行，而不是人的意志。

Computers are good at following instructions, but not at reading your mind.

——唐纳德·克努特（Donald Knuth）

软件算法和程序设计技术的先驱者

排版软件 Tex 的发明人，《计算机程序设计的艺术》巨作的作者

学习目标

（1）了解 3 类基本组合数据类型。
（2）理解列表概念并掌握 Python 中列表的使用。
（3）理解字典概念并掌握 Python 中字典的使用。
（4）运用列表管理采集的信息，构建数据结构。
（5）运用字典处理复杂的数据信息。
（6）运用组合数据类型进行文本词频统计。

在中国文学史上，《三国演义》毫无疑问是影响力最大的小说之一。书中的主要人物家喻户晓，智慧的代言人诸葛亮、红脸的关公和白脸的曹操……这些历史人物的形象定位，几乎都来自于《三国演义》。该书中数百个栩栩如生的角色，究竟谁才是罗贯中的主角？有人说是刘备，有人说是诸葛亮，更多的人则比较偏向曹操。

空口无凭，一起来编个程序统计《三国演义》人物的出场次数吧。

6.1 组合数据类型概述

> **要点：** 组合数据类型为多个同类型或不同类型数据提供单一表示。组合数据类型分 3 类：序列类型、集合类型和映射类型。

计算机不仅对单个变量表示的数据进行处理，更通常的情况是，计算机需要对一组数据进行批量处理。例如：

（1）给定一组单词{python, data, function, list, loop}，计算并输出每个单词的长度。

（2）给定一个学院的学生信息，统计男女生比例。

（3）一次实验产生了很多组数据，对这些大量数据进行分析。

以单词统计问题为例，在计算一个单词长度之前，程序需要使用一个变量表示这个单词，对于一组单词，需要很多个变量。有两个解决方案：为每个单词分配一个变量，从变量命名上加以区分，例如，a01、a02 分别存储第一个、第二个元素；或者采用一个数据结构存储这组数据，对每个元素采用索引加以区分，例如 a 表示这组元素，a[0]为该组第一个元素，a[1]为第二个元素。两个方案哪个更好呢？显然，第二个方案更好。假定单词数量是 500 个而不是 5 个，使用第一种方法将是灾难。此外，对每个元素单独定义变量，不利于循环操作。

第 3 章介绍了数字类型，包括整数类型、浮点数类型和复数类型，这些类型仅能表示一个数据，这种表示单一数据的类型称为基本数据类型。然而，实际计算中却存在大量同时处理多个数据的情况，这需要将多个数据有效组织起来并统一表示，这种能够表示多个数据的类型称为组合数据类型。

组合数据类型能够将多个同类型或不同类型的数据组织起来，通过单一的表示使数据操作更有序、更容易。根据数据之间的关系，组合数据类型可以分为 3 类：序列类型、集合类型和映射类型。

序列类型是一个元素向量，元素之间存在先后关系，通过序号访问，元素之间不排他。

集合类型是一个元素集合，元素之间无序，相同元素在集合中唯一存在。

映射类型是“键-值”数据项的组合，每个元素是一个键值对，表示为(key, value)。

在 Python 中，每一类组合数据类型都对应一个或多个具体的数据类型，结合本书章节安排，组合数据类型的分类构成如图 6.1 所示，其中加粗字体表示 Python 支持的具体数据类型。

6.1.1 序列类型

序列类型是一维元素向量，元素之间存在先后关系，通过序号访问。序列的基

本思想和表示方法均来源于数学概念。在数学中，经常给每个序列一个名字，例如，n 个数的序列 S，可以表示如下：

$$S=s_0, s_1, s_2, \ldots, s_{n-1}$$

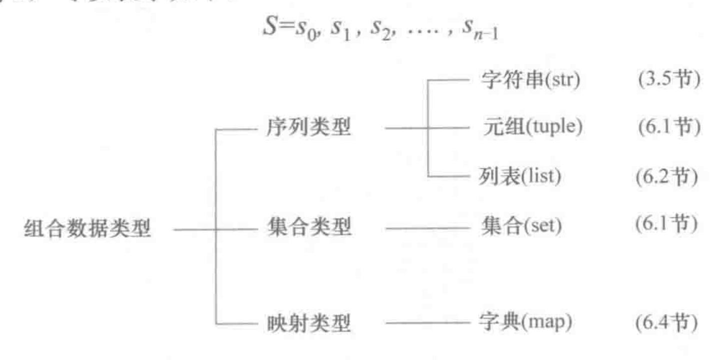


图 6.1　组合数据类型的分类和章节索引

当需要访问序列中某个特定值时，只需要通过下标标出即可。例如，需要找到第 2 个元素，即可通过 s_2 获得。这种采用集合名字和下标相结合的表示方法可以简洁地表示序列运算，例如，对上述序列 S 求和可以表示如下：

$$\sum_{i=0}^{n-1} S_i$$

由于元素之间存在顺序关系，所以序列中可以存在数值相同但位置不同的元素。序列类型支持成员关系操作符（in）、长度计算函数（len()）、分片（[]），元素本身也可以是序列类型。

Python 语言中有很多数据类型都是序列类型，其中比较重要的是 str（字符串）、tuple（元组）和 list（列表）。字符串（str）可以看成是单一字符的有序组合，属于序列类型。同时，由于字符串类型十分常用且单一字符串只表达一个含义，也被看作是基本数据类型。元组是包含 0 个或多个数据项的不可变序列类型。元组生成后是固定的，其中任何数据项不能替换或删除。列表则是一个可以修改数据项的序列类型，使用也最灵活，将在 6.2 节详细介绍。无论哪种具体数据类型，只要它是序列类型，都可以使用相同的索引体系，即正向递增序号和反向递减序号，如图 6.2 所示。

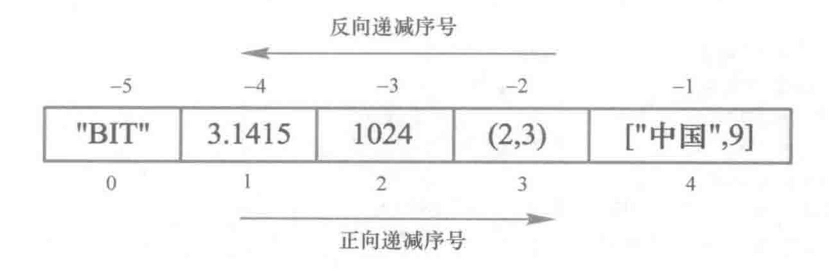


图 6.2　序列类型的索引体系

序列类型有 12 个通用的操作符和函数，如表 6.1 所示。

表 6.1 序列类型的通用操作符和函数（共 12 个）

操 作 符	描 述
x in s	如果 x 是 s 的元素，返回 True，否则返回 False
x not in s	如果 x 不是 s 的元素，返回 True，否则返回 False
s + t	连接 s 和 t
s * n 或 n * s	将序列 s 复制 n 次
s[i]	索引，返回序列的第 i 个元素
s[i: j]	分片，返回包含序列 s 第 i 到 j 个元素的子序列（不包含第 j 个元素）
s[i: j: k]	步骤分片，返回包含序列 s 第 i 到 j 个元素以 k 为步数的子序列
len(s)	序列 s 的元素个数（长度）
min(s)	序列 s 中的最小元素
max(s)	序列 s 中的最大元素
s.index(x[, i[, j]])	序列 s 中从 i 开始到 j 位置中第一次出现元素 x 的位置
s.count(x)	序列 s 中出现 x 的总次数

元组（tuple）是序列类型中比较特殊的类型，因为它一旦创建就不能被修改。元组类型在表达固定数据项、函数多返回值、多变量同步赋值、循环遍历等情况下十分有用。Python 中元组采用逗号和圆括号（可选）来表示，例如：

```
>>>creature = "cat", "dog", "tiger", "human"
>>>creature
('cat', 'dog', 'tiger', 'human')
>>>color = ("red", 0x001100, "blue", creature)
>>>color
('red', 4352, 'blue', ('cat', 'dog', 'tiger', 'human'))
>>>color[2]
'blue'
>>>color[-1][2]
'tiger'
```

生成元组只需要使用逗号将元素隔离开即可，例如上例中的元组 creature，也可以增加圆括号，但圆括号在不混淆语义的情况下不是必需的。一个元组可以作为另一个元组的元素，可以采用多级索引获取信息，例如元组 color 中包含了元组 creature，可以用 color[-1][2]获取对应元素值。元组除了用于表达固定数据项外，还常用于如下 3 种情况：函数多返回值、多变量同步赋值、循环遍历，例如：

```
>>>def func(x):  #函数多返回值
       return x, x**3
>>>a, b = 'dog', 'tiger'  #多变量同步赋值
>>>a, b = (b, a)          #多变量同步赋值，括号可省略
>>>import math
>>>for x, y in ((1,0), (2,5), (3,8)):  #循环遍历
       print(math.hypot(x,y))          #求多个坐标值到原点的距离
```

6.1.2 集合类型

集合类型与数学中集合的概念一致，即包含 0 个或多个数据项的无序组合。集

合中的元素不可重复，元素类型只能是固定数据类型，例如整数、浮点数、字符串、元组等，列表、字典和集合类型本身都是可变数据类型，不能作为集合的元素出现。Python 编译器中界定固定数据类型与否主要考察类型是否能够进行哈希运算。能够进行哈希运算的类型都可以作为集合元素。Python 提供了一种同名的具体数据类型——集合（set）。

拓展： 哈希运算

哈希运算可以将任意长度的二进制值映射为较短的固定长度的二进制值，这个小的二进制值称为哈希值。哈希值是对数据的一种有损且紧凑的表示形式。Python 提供了一个内置的哈希运算函数 hash()，它可以对大多数数据类型产生一个哈希值，例如：

```
>>>hash("PYTHON")
-924726778
>>>hash("IS")
1964309299
>>> hash("GOOD")
1808989930
>>>hash("PYTHON IS GOOD")
-775775176
```

这些哈希值与哈希前的内容无关，也和这些内容的组合无关。可以说，哈希是数据在另一个数据维度的体现。

由于集合是无序组合，它没有索引和位置的概念，不能分片，集合中元素可以动态增加或删除。集合用大括号（{}）表示，可以用赋值语句生成一个集合，例如：

```
>>>S = {425, "BIT", (10, "CS"), 424}
>>>S
{424, 425, (10, 'CS'), 'BIT'}
>>>T = {425, "BIT", (10, "CS"), 424, 425, "BIT"}
>>>T
{424, 425, (10, 'CS'), 'BIT'}
```

从上例可以看到，由于集合元素是无序的，集合的打印效果与定义顺序可以不一致。由于集合元素独一无二，使用集合类型能够过滤掉重复元素。

set(x)函数可以用于生成集合，输入的参数可以是任何组合数据类型，返回结果是一个无重复且排序任意的集合，例如：

```
>>>W = set("apple")
{'e', 'p', 'a', 'l'}
>>>V = set(("cat", "dog", "tiger", "human"))
{'cat', 'human', 'dog', 'tiger'}
```

集合类型有 10 个操作符，如表 6.2 所示。

表 6.2 集合类型的操作符（共 10 个）

<table>
<tr><th>操 作 符</th><th>描 述</th></tr>
<tr><td>S – T 或 S.difference(T)</td><td>返回一个新集合，包括在集合 S 中但不在集合 T 中的元素</td></tr>
<tr><td>S-=T 或 S.difference_update(T)</td><td>更新集合 S，包括在集合 S 中但不在集合 T 中的元素</td></tr>
<tr><td>S & T 或 S.intersection(T)</td><td>返回一个新集合，包括同时在集合 S 和 T 中的元素</td></tr>
<tr><td>S&=T 或 S.intersection_update(T)</td><td>更新集合 S，包括同时在集合 S 和 T 中的元素</td></tr>
<tr><td>S^T 或 s.symmetric_difference(T)</td><td>返回一个新集合，包括集合 S 和 T 中的元素，但不包括同时在其中的元素</td></tr>
<tr><td>S=^T 或 s.symmetric_difference_update(T)</td><td>更新集合 S，包括集合 S 和 T 中的元素，但不包括同时在其中的元素</td></tr>
<tr><td>S|T 或 S.union(T)</td><td>返回一个新集合，包括集合 S 和 T 中的所有元素</td></tr>
<tr><td>S=|T 或 S.update(T)</td><td>更新集合 S，包括集合 S 和 T 中的所有元素</td></tr>
<tr><td>S<=T 或 S.issubset(T)</td><td>如果 S 与 T 相同或 S 是 T 的子集，返回 True，否则返回 False，可以用 S<T 判断 S 是否是 T 的真子集</td></tr>
<tr><td>S>=T 或 S.issuperset(T)</td><td>如果 S 与 T 相同或 S 是 T 的超集，返回 True，否则返回 False，可以用 S>T 判断 S 是否是 T 的真超集</td></tr>
</table>

上述操作符表达了集合类型的 4 种基本操作：交集（&）、并集（|）、差集（-）、补集（^），操作逻辑与数学定义相同，如图 6.3 所示。

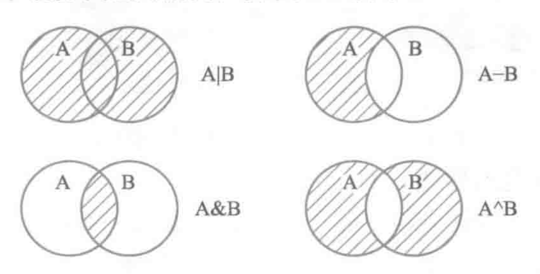


图 6.3 集合类型的 4 种基本操作

集合类型有 10 个操作函数或方法，如表 6.3 所示。

表 6.3 集合类型的操作函数或方法（共 10 个）

操作函数或方法	描 述
S.add(x)	如果数据项 x 不在集合 S 中，将 x 增加到 s
S.clear()	移除 S 中的所有数据项
S.copy()	返回集合 S 的一个副本
S.pop()	随机返回集合 S 中的一个元素，如果 S 为空，产生 KeyError 异常
S.discard(x)	如果 x 在集合 S 中，移除该元素；如果 x 不在集合 S 中，不报错
S.remove(x)	如果 x 在集合 S 中，移除该元素；不在则产生 KeyError 异常
S.isdisjoint(T)	如果集合 S 与 T 没有相同元素，返回 True
len(S)	返回集合 S 的元素个数
x in S	如果 x 是 S 的元素，返回 True，否则返回 False
x not in S	如果 x 不是 S 的元素，返回 True，否则返回 False

集合类型主要用于 3 个场景：成员关系测试、元素去重和删除数据项，例如：

```
>>>"BIT" in {"PYTHON", "BIT", 123, "GOOD"} #成员关系测试
True
>>>tup = ("PYTHON", "BIT", 123, "GOOD", 123) #元素去重
>>>set(tup)
{123, 'GOOD', 'BIT', 'PYTHON'}
>>>newtup = tuple(set(tup)-{'PYTHON'}) # 去重同时删除数据项
('GOOD', 123, 'BIT')
```

集合类型与其他类型最大的不同在于它不包含重复元素，因此，当需要对一维数据进行去重或进行数据重复处理时，一般通过集合来完成。

6.1.3 映射类型

映射类型是“键-值”数据项的组合，每个元素是一个键值对，即元素是(key, value)，元素之间是无序的。键值对(key, value)是一种二元关系，源于属性和值的映射关系，对应实例如图 6.4 所示。

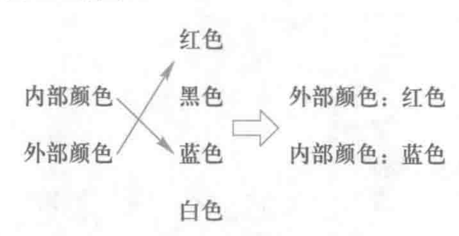


图 6.4 映射关系和键值对实例

键（key）表示一个属性，也可以理解为一个类别或项目，值（value）是属性的内容，键值对刻画了一个属性和它的值。键值对将映射关系结构化，用于存储和表达。在 Python 中，映射类型主要以字典（dict）体现。6.4.1 节将进一步介绍字典类型。

思考与练习

6.1 请比较元组和集合的区别，思考如何实现元组和集合的互相转换？

6.2 两个集合：S1={1,3,5,6}，S2={2,5,6}，请计算 S1|S2、S1&S2、S1^S2 和 S1-S2 的值。

6.3 思考序列、集合和映射在数据关系层面的含义。

6.2 列表类型和操作

要点： 列表是包含 0 个或多个对象引用的有序序列，没有长度限制，可自由增删元素，使用灵活。

6.2.1 列表类型的概念

列表（list）是包含 0 个或多个对象引用的有序序列，属于序列类型。与元组不同，列表的长度和内容都是可变的，可自由对列表中的数据项进行增加、删除或替换。列表没有长度限制，元素类型可以不同，使用非常灵活。

由于列表属于序列类型，所以列表也支持成员关系操作符（in）、长度计算函数（len()）、分片（[]）。列表可以同时使用正向递增序号和反向递减序号，可以采用标准的比较操作符（<、<=、==、!=、>=、>）进行比较，列表的比较实际上是单个数据项的逐个比较。

列表用中括号（[]）表示，也可以通过 list()函数将元组或字符串转化成列表。直接使用 list()函数会返回一个空列表，例如：

```
>>>ls = [425, "BIT", [10, "CS"], 425]
>>>ls
[425, 'BIT', [10, 'CS'], 425]
>>>ls[2][-1][0]
'C'
>>>list((425, "BIT", [10, "CS"], 425))
[425, 'BIT', [10, 'CS'], 425]
>>>list("中国是一个伟大的国家")
['中', '国', '是', '一', '个', '伟', '大', '的', '国', '家']
>>>list()
[]
```

拓展： 列表和数组

每种编程语言都提供一个或多个表示一组元素的方法，例如，C 语言采用数组，Python 采用列表。在大多数语言中，数组十分常见，仅有少量语言采用列表而不是数组。列表和数组类似，但并不完全一样，列表和数组有两个显著不同。

第一，数组需要预先分配大小，列表则不需要。当创建一个数组时，必须指定数组的大小，即它能容纳元素的个数。如果不知道有多少元素，必须假设一个最大可能的数值，再按照这个最大值分配一个数组，并记录数组中实际存储元素的个数，以保证实际使用的元素数量不超过数组的限制。列表则没有预分配大小的要求和限制，创建列表变量时不需要知道元素个数，可以在使用中动态插入任何数量的元素。

第二，数组要求元素类型一致，列表则不需要。数组要求每个元素具有相同的数据类型，如果某个元素是整数，则数组中全部元素都是整数。列表没有上述限制，列表中不同元素的类型可以相同，也可以不同，甚至，列表中的元素也可以是列表类型。列表的这个特点十分灵活，为程序编写提供了很大的设计空间。

与整数和字符串不同，列表要处理一组数据，因此，列表必须通过显式的数据

赋值才能生成，简单将一个列表赋值给另一个列表不会生成新的列表对象，例如：

```
>>>ls = [425, "BIT", 1024]  #用数据赋值产生列表 ls
>>>lt = ls        #lt 是 ls 所对应数据的引用，lt 并不包含真实数据
>>>ls[0] = 0
>>>lt
[0, 'BIT', 1024]
```

如上例所示，ls 由实际数据赋值产生，为列表对象。将 ls 赋值给列表 lt 仅能产生对列表 ls 的一个新的引用，此时，lt 和 ls 变量都是实际数据[425,"BIT", 1024]的表示或引用，真实数据只存储一份，因此，修改 ls 也同时修改了 lt，这个关系如图 6.5 所示。

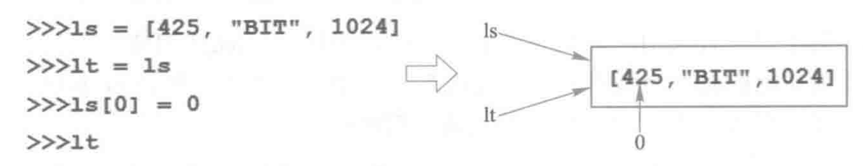


图 6.5　列表对象的创建和引用

6.2.2　列表类型的操作

列表是序列类型，因此，表 6.1 中 12 个序列类型的操作符和函数都可应用于列表类型。由于列表是可变的，表 6.4 给出了列表类型额外的 14 个常用函数或方法。

表 6.4　列表类型特有的函数或方法（共 14 个）

函数或方法	描　述
ls[i] = x	替换列表 ls 第 i 数据项为 x
ls[i: j] = lt	用列表 lt 替换列表 ls 中第 i 到第 j 项数据（不含第 j 项，下同）
ls[i: j: k] = lt	用列表 lt 替换列表 ls 中第 i 到第 j 项以 k 为步数的数据
del ls[i: j]	删除列表 ls 第 i 到第 j 项数据，等价于 ls[i: j]=[]
del ls[i: j: k]	删除列表 ls 第 i 到第 j 项以 k 为步数的数据
ls+=lt 或 ls.extend(lt)	将列表 lt 元素增加到列表 ls 中
ls *= n	更新列表 ls，其元素重复 n 次
ls.append(x)	在列表 ls 最后增加一个元素 x
ls.clear()	删除 ls 中的所有元素
ls.copy()	生成一个新列表，复制 ls 中的所有元素
ls.insert(i, x)	在列表 ls 的第 i 位置增加元素 x
ls.pop(i)	将列表 ls 中的第 i 项元素取出并删除该元素
ls.remove(x)	将列表中出现的第一个元素 x 删除
ls.reverse(x)	列表 ls 中的元素反转

上述操作符主要处理列表的增删改等功能，例如：

```
>>>vlist = list(range(5))
```

```
>>>vlist
[0, 1, 2, 3, 4]
>>>len(vlist[2:])        #计算从第 3 个位置开始到结尾的子串长度
3
>>>2 in vlist            #判断 2 是否在列表 vlist 中
True
>>>vlist[3]="python"     #修改序号 3 的元素值和类型
>>>vlist
[0, 1, 2, 'python', 4]
>>>vlist[1:3]=["bit", "computer"]
>>>vlist
[0, 'bit', 'computer', 'python', 4]
```

上述例子中，vlist[3]从整数变成了字符串，子序列 vlist[1:3]被另一个列表赋值修改。需要注意的是，当使用一个列表改变另一个列表值时，Python 不要求两个列表长度一样，但遵循“多增少减”的原则，例如:

```
>>>vlist[1:3]=["new_bit", "new_computer", 123]
>>>vlist
[0, 'new_bit', 'new_computer', 123, 'python', 4]
>>>vlist[1:3]=["fewer"]
>>>vlist
[0, 'fewer', 123, 'python', 4]
```

vlist[1:3]子序列包含两个元素，对其赋值时却给了 3 个元素，Python 接受这种方式，并不会报错，vlist 结果包含了赋值列表中的多余元素。同样，当使用包含更少元素赋值列表时，原列表元素会相应减少。可以通过赋给更多或更少元素实现对列表元素的插入或删除。

阶段测试 6-1：Python 列表使用小测验



与元组一样，列表可以通过 for-in 语句对其元素进行遍历，基本语法结构如下：

```
for <任意变量名>  in  <列表名>:
        <语句块>
```

```
>>>for e in vlist:
       print(e, end=" ")
0 fewer 123 python 4
```

列表是一个十分灵活的数据结构，它具有处理任意长度、混合类型数据的能力，并提供了丰富的基础操作符和方法。当程序需要使用组合数据类型管理批量数据时，请尽量使用列表类型。

程序练习 6-1：半小时学 Python 列表



思考与练习

6.4 列表 ls=[2,5,7,1,6]，请对列表按照升序和降序的方式分别排列。提示：请使用 Python 内置函数。

6.5 列表 ls1=[30,1,2,0]，ls2=[1,21,133]，请比较两个列表。

6.6　列表 ls1=[1,43]，ls2=ls1，ls1[0]=22，请计算并思考两个列表的运算结果。

6.7　列表 ls=[[2,3,7],[[3,5],25],[0,9]]，len(ls)值是多少？

6.3　实例 9：基本统计值计算

要点： 这是一个计算多数据基本统计值的例子。

统计是计算科学、管理学、社会学、数学等诸多领域的基本问题，相关问题、方法和技术组成了一门学科，即“统计学”。Python 的列表数据结构能够支持基本的数据统计应用。本节以最简单的统计问题为例，求解一组不定长数据的基本统计值，即平均值、标准差、中位数。

一组数据表示为 $S=s_0, s_1, \cdots, s_{n-1}$，其算术平均值、标准差分别表示如下：

$$m=(\sum_{i=0}^{n-1} s_i)/n$$

$$d=\sqrt{(\sum_{i=0}^{n-1}(s_i-m)^2)/(n-1)}$$

中位数指 S 中所有数按照从小到大（或者从大到小）顺序排列后，处于最中间位置的数据值。如果 n 是奇数，则序列 S 的最中间位置是一个数据，可以表示为 $s_{n//2}(n=0,1,2,\cdots)$；如果 n 是偶数，序列 S 不存在一个最中间位置，则中位数表示为最中间两个位置数据的平均值，即$(s_{n//2-1}+s_{n//2})/2$。例如，（5,2,1,3,4）的中位数是 3，而（4,2,1,3）的中位数是(2+3)/2 为 2.5。这个问题的 IPO 描述如下。

输入：从用户输入、文件、网络等途径获取一组数据

处理：适当的数据结构和算法

输出：平均值、标准差和中位数

由于平均值、标准差和中位数是 3 个不同的计算目标，使用函数方式编写计算程序。定义 getNum()函数从用户输入获得数据，mean()函数计算平均值，dev()函数计算标准差，median()函数计算中位数。由于该问题不限制用户输入数据的最大个数，所以，使用列表作为承载和存储数据的数据类型。实例代码 9.1 的全部代码如下：

源代码 6-1：基本统计值计算



实例代码 9.1　　e9.1CalStatistics.py

```
1  #e9.1CalStatistics.py
2  from math import sqrt
3  def getNum():        #获取用户输入
4      nums = []
5      iNumStr = input("请输入数字(直接输入回车退出): ")
6      while iNumStr != "":
7          nums.append(eval(iNumStr))
```

```
8          iNumStr = input("请输入数字(直接输入回车退出): ")
9       return nums
10  def mean(numbers):  #计算平均值
11      s = 0.0
12      for num in numbers:
13          s = s + num
14      return s / len(numbers)
15  def dev(numbers, mean): #计算方差
16      sdev = 0.0
17      for num in numbers:
18          sdev = sdev + (num - mean)**2
19      return sqrt(sdev / (len(numbers)-1))
20  def median(numbers):    #计算中位数
21      sorted(numbers)
22      size = len(numbers)
23      if size % 2 == 0:
24          med = (numbers[size//2-1] + numbers[size//2])/2
25      else:
26          med = numbers[size//2]
27      return med
28  n =  getNum()  #主体函数
29  m =  mean(n)
30  print("平均值:{},方差:{:.2},中位数:{}.".format(m,\
            dev(n,m),median(n)))
```

该程序运行结果如下：

```
>>>
请输入数字(直接输入回车退出): 99
请输入数字(直接输入回车退出): 98
请输入数字(直接输入回车退出): 97
请输入数字(直接输入回车退出): 96
请输入数字(直接输入回车退出): 95
请输入数字(直接输入回车退出):
平均值:97.0,方差:1.6,中位数:97.
```

程序整体从第 28 行开始执行，先后调用 getNum()、mean()、dev()和 median()函数。利用函数的模块化设计能够复用代码并增加代码的可读性。每个函数内部都采用了简单的语句。

getNum()函数循环从控制台获得用户输入的数字，当用户按 Enter 键时退出，所有数据保存在 nums 列表中。列表 nums 初始化时定义为空，而后根据输入逐渐增加其长度。

mean()函数用浮点数 s 记录列表 numbers 求和的结果。其中，for 语句表示从列表 numbers 中取出每一个元素，将其加到 s 变量中，直到 numbers 中的最后一个元

素。最后，通过 return 语句返回平均值，len(numbers)用于计算列表的长度。

为了计算标准差，需要知道数据的平均值，由于 mean()函数已经可以计算平均值，将均值作为一个参数输入标准差 dev()函数。dev()函数中(val)**2 用于计算 val 的平方，sqrt(val)计算 val 的平方根。

根据中位数的定义，中位数函数 median()首先使用 Python 内置函数 sorted()对列表 numbers 进行排序，然后根据中位数定义计算中位数。

列表在实现基本数据统计时发挥了很重要的作用，主要表现在以下 3 个方面。

（1）列表是一个动态长度的数据结构，可以根据需求增加或减少元素。

（2）列表的一系列方法或操作符为计算提供了简单的元素运算手段。

（3）列表提供了对每个元素的简单访问方式及所有元素的遍历方式。

拓展： 中位数的含义

中位数是统计学中常用的一个指标，它可以将数值集合划分为相等的两部分。中位数不受数列分布的极大或极小值影响，从而在一定程度上代表了分布数列，它也是一种衡量集中趋势的方法。

根据国家统计局发布的 2016 年上半年主要经济指标，上半年全国居民人均可支配收入 11 886 元，全国居民人均可支配收入中位数 10 505 元，说明全国多于一半的居民收入低于平均值。

思考与练习

6.8　请修改实例代码 9.1 中的第 21 行，使用 sorted()函数实现数据的降序排列。

6.9　请在实例代码 9.1 中增加函数，实现最大值、最小值的计算和输出。

6.10　请阅读国家统计局发布的某份统计报告，思考中位数及其他统计指标的含义。

6.4　字典类型和操作

要点： 字典是包含 0 个或多个键值对的集合，没有长度限制，可以根据键索引值的内容。

6.4.1　字典类型的概念

列表是存储和检索数据的有序序列。当访问列表中的元素时，可以通过整数的索引来查找它，这个索引是元素在列表中的序号，列表的索引模式是“<整数序号>查找<被索引内容>”。

很多应用程序需要更灵活的信息查找方式，例如，在检索学生或员工信息时，

需要基于身份证号码进行查找，而不是信息存储的序号。在编程术语中，根据一个信息查找另一个信息的方式构成了“键值对”，它表示索引用的键和对应的值构成的成对关系，即通过一个特定的键（身份证号码）来访问值（学生信息）。实际应用中有很多“键值对”的例子，例如，姓名和电话号码、用户名和密码、邮政编码和运输成本、国家名称和首都等。由于键不是序号，无法使用列表类型进行有效存储和索引。

通过任意键信息查找一组数据中值信息的过程叫映射，Python 语言中通过字典实现映射。Python 语言中的字典可以通过大括号（{}）建立，建立模式如下：

```
{<键 1>:<值 1>, <键 2>:<值 2>, … , <键 n>:<值 n>}
```

其中，键和值通过冒号连接，不同键值对通过逗号隔开。从 Python 设计角度考虑，由于大括号{}可以表示集合，因此字典类型也具有和集合类似的性质，即键值对之间没有顺序且不能重复。简单说，可以把字典看成元素是键值对的集合。下面是一个简单的字典，它存储国家和首都的键值对：

```
>>>Dcountry={"中国":"北京", "美国":"华盛顿", "法国":"巴黎"}
>>>print(Dcountry)
{'中国': '北京', '法国': '巴黎', '美国': '华盛顿'}
```

注意到，字典打印出来的顺序与创建之初的顺序不同，这不是错误。字典是集合类型的延续，所以各个元素并没有顺序之分。如果想保持一个集合中元素的顺序，需要使用列表，而不是字典。

字典最主要的用法是查找与特定键相对应的值，这通过索引符号来实现。例如：

```
>>>Dcountry["中国"]
'北京'
```

一般来说，字典中键值对的访问模式如下，采用中括号格式：

```
<值> = <字典变量>[<键>]
```

字典中对某个键值的修改可以通过中括号的访问和赋值实现，例如：

```
>>>Dcountry["中国"]='大北京'
>>>print(Dcountry)
{'中国': '大北京', '法国': '巴黎', '美国': '华盛顿'}
```

总结起来，字典是存储可变数量键值对的数据结构，键和值可以是任意数据类型，包括程序自定义的类型。Python 字典效率非常高，甚至可以存储几十万项内容。

拓展：索引

索引是按照一定顺序检索内容的体系。编程语言的索引主要包括两类：数字索引，也称为位置索引；字符索引，也称为单词索引。数字索引采用数字作为索引数的方法，可以通过整数序号找到内容。字符索引采用字符作为索引词，通过具体的索引词找到数据，例如，现实生活中的汉语词典，通过汉语词找到释义。Python 语言中，字符串、列表、元组等都采用数字索引，字典采用字符索引。

6.4.2　字典类型的操作

与列表相似，Python 字典也有非常灵活的操作方法。使用大括号可以创建字典，并指定初始值，通过中括号可以增加新的元素，例如：

```
>>>Dcountry={"中国":"北京", "美国":"华盛顿", "法国":"巴黎"}
>>>Dcountry["英国"]="伦敦"
>>>print(Dcountry)
{'中国': '北京', '法国': '巴黎', '美国': '华盛顿', '英国': '伦敦'}
```

直接使用大括号（{}）可以创建一个空的字典，并通过中括号（[]）向其增加元素，例如：

```
>>>Dp={}
>>>Dp['2^10']=1024
>>>print(Dp)
{'2^10': 1024}
```

需要注意的是，尽管集合类型也用大括号表示，直接使用大括号（{}）生成一个空的字典，而不是集合。生成空集合需要使用函数 set()。

字典在 Python 内部也已采用面向对象方式实现，因此也有一些对应的方法，采用<a>.<b>()格式，此外，还有一些函数能够用于操作字典，这些函数和方法如表 6.5 所示。

表 6.5　字典类型的函数和方法（共 9 个）

函数和方法	描　述
<d>.keys()	返回所有的键信息
<d>.values()	返回所有的值信息
<d>.items()	返回所有的键值对
<d>.get(<key>,<default>)	键存在则返回相应值，否则返回默认值
<d>.pop(<key>,<default>)	键存在则返回相应值，同时删除键值对，否则返回默认值
<d>.popitem()	随机从字典中取出一个键值对，以元组(key, value)形式返回
<d>.clear()	删除所有的键值对
del <d>[<key>]	删除字典中某一个键值对
<key> in <d>	如果键在字典中则返回 True，否则返回 False

上述方法的一些例子如下，如果希望 keys()、values()和 items()方法返回列表类型，可以采用 list()函数将返回值转换成列表。

```
>>>Dcountry={"中国":"北京", "美国":"华盛顿", "法国":"巴黎"}
>>>Dcountry.keys()
dict_keys(['中国', '美国', '法国'])
>>>list(Dcountry.values())
['北京', '华盛顿', '巴黎']
```

```
>>>Dcountry.items()
dict_items([('中国', '北京'), ('美国', '华盛顿'), ('法国', '巴黎')])
>>>'中国' in Dcountry   #只对键进行判断
True
>>>Dcountry.get('美国', '悉尼') #'美国'在字典中存在
'华盛顿'
>>>Dcountry.get('澳大利亚', '悉尼') #'澳大利亚'在字典中不存在
'悉尼'
```

与其他组合类型一样，字典可以通过 for-in 语句对其元素进行遍历，基本语法结构如下：

```
for <变量名>  in  <字典名>:
    <语句块>
```

由于键值对中的键相当于索引，因此，for 循环返回的变量名是字典的索引值。如果需要获得键对应的值，可以在语句块中通过 get()方法获得。

```
>>>for key in Dcountry:
       print(key)
中国
美国
法国
```

字典是实现键值对映射的数据结构，它采用固定数据类型的键数据作为索引，十分灵活，具有处理任意长度、混合类型键值对的能力。为了更好地认识和使用字典，请理解如下一些基本原则。

阶段测试 6-2：Python 字典使用小测验



（1）字典是一个键值对的集合，该集合以键为索引，一个键信息只对应一个值信息。

（2）字典中元素以键信息为索引访问。

（3）字典长度是可变的，可以通过对键信息赋值实现增加或修改键值对。

思考与练习

程序练习 6-2：半小时学 Python 字典

6.11 判断题：在字典里，同一个键可以对应两个或多个值。

6.12 字典 D={"张三":88,"李四":90,"王五":73,"赵六":82}，写出下列操作的代码。

（1）向字典中添加键值对“"钱七":90”。

（2）修改"王五"对应的值为 93。

（3）删除"赵六"对应的键值对。

6.13 下面是正确的字典创建方式的是（　　）。

A. d={1:[1,2],3:[3,4]}　　B. d={[1,2]:1,[3,4]:3}

C. d={(1,2):1,(3,4):3}　　D. d={1: "张三 ", 3: "李四 "}

E. d={ "张三 ":1, "李四 ":3}

6.14 对于字典 d={"abc":1, "qwe":3, "zxc":2}，len(d)的结果是多少？

6.5　模块 4：jieba 库的使用

要点：jieba 是 Python 中一个重要的第三方中文分词函数库。

6.5.1　jieba 库概述

对于一段英文文本，例如"China is a great country"，如果希望提取其中的单词，只需要使用字符串处理的 split()方法即可，例如：

```
>>>"China is a great country".split()
['China', 'is', 'a', 'great', 'country']
```

然而，对于一段中文文本，例如，"中国是一个伟大的国家"，获得其中的单词（不是字符）十分困难，因为英文文本可以通过空格或者标点符号分隔，而中文单词之间缺少分隔符，这是中文及类似语言独有的“分词”问题。上例中，分词能够将"中国是一个伟大的国家"分为"中国"、"是"、"一个"、"伟大"、"的"、"国家"等一系列词语。

jieba（“结巴”）是 Python 中一个重要的第三方中文分词函数库，例如：

```
>>>import jieba
>>>jieba.lcut("中国是一个伟大的国家")
['中国', '是', '一个', '伟大', '的', '国家']
```

jieba 库是第三方库，不是 Python 安装包自带的，因此，需要通过 pip 指令安装，具体安装方法请参考 8.6 节。pip 安装命令如下：

```
:\>pip install jieba   # 或者 pip3 install jieba
```

jieba 库的分词原理是利用一个中文词库，将待分词的内容与分词词库进行比对，通过图结构和动态规划方法找到最大概率的词组。除了分词，jieba 还提供增加自定义中文单词的功能。

图片资料 6-1：Python 快速参考之 random、jieba、datetime 库



jieba 库支持 3 种分词模式：精确模式，将句子最精确地切开，适合文本分析；全模式，把句子中所有可以成词的词语都扫描出来，速度非常快，但是不能消除歧义；搜索引擎模式，在精确模式的基础上，对长词再次切分，提高召回率，适合用于搜索引擎分词。

6.5.2　jieba 库解析

jieba 库主要提供分词功能，可以辅助自定义分词词典。jieba 库中包含的主要函

数如表 6.6 所示。

表 6.6 jieba 库常用的分词函数（共 7 个）

函 数	描 述
jieba.cut(s)	精确模式，返回一个可迭代的数据类型
jieba.cut(s, cut_all=True)	全模式，输出文本 s 中所有可能的单词
jieba.cut_for_search(s)	搜索引擎模式，适合搜索引擎建立索引的分词结果
jieba.lcut(s)	精确模式，返回一个列表类型，建议使用
jieba.lcut(s, cut_all=True)	全模式，返回一个列表类型，建议使用
jieba.lcut_for_search(s)	搜索引擎模式，返回一个列表类型，建议使用
jieba.add_word(w)	向分词词典中增加新词 w

针对上述分词函数，举例如下：

```
>>>import jieba
>>>jieba.lcut("中华人民共和国是一个伟大的国家")
['中华人民共和国', '是', '一个', '伟大', '的', '国家']
>>>jieba.lcut("中华人民共和国是一个伟大的国家", cut_all=True)
['中华', '中华人民', '中华人民共和国', '华人', '人民', '人民共和国', '共和',
'共和国', '国是', '一个', '伟大', '的', '国家']
>>>jieba.lcut_for_search("中华人民共和国是一个伟大的国家")
['中华', '华人', '人民', '共和', '共和国', '中华人民共和国', '是', '一个',
'伟大', '的', '国家']
```

jieba.lcut()函数返回精确模式，输出的分词能够完整且不多余地组成原始文本；jieba.lcut(,True)函数返回全模式，输出原始文本中可能产生的所有问题，冗余性最大；jieba.lcut_for_search()函数返回搜索引擎模式，该模式首先执行精确模式，然后再对其中的长词进一步切分获得结果。由于列表类型通用且灵活，建议读者使用上述 3 个能够返回列表类型的分词函数。

默认情况下，表 6.6 中的 jieba.cut()等 6 个分词函数能够较高概率识别自定义的新词，比如名字或缩写，例如，下例中本书作者的姓名不在词典中，但分词函数能够根据中文字符间的相关性识别为一个词。对于无法识别的分词，也可以通过 jieba.add_word()函数向分词库添加，例如：

```
>>>import jieba
>>>jieba.lcut("嵩天老师在努力教学 Python 语言")
['嵩天', '老师', '在', '努力', '教学', 'Python', '语言']
>>> jieba.lcut("习大大期盼有更好的教育")
['习', '大大', '期盼', '有', '更好', '的', '教育']
>>>jieba.add_word("习大大")
>>> jieba.lcut("习大大期盼有更好的教育")
['习大大', '期盼', '有', '更好', '的', '教育']
```

jieba 库还有更丰富的分词功能，这涉及自然语言处理领域，本节不再深入介绍。掌握表 6.6 中 7 个分词函数能够处理绝大部分与中文文本相关的分词问题。

拓展： 第三方库

Python 语言的第三方库指不在 Python 安装包中的函数库，也是非标准函数库。这类函数库一般由全球各领域专业人士结合专业特点和兴趣开发。Python 语言构建了一个开放和自由的生态环境，对第三方库的开发没有强制要求，因此，Python 语言的第三方库发展十分迅速。截至 2016 年 9 月，Python 官方网站注册的第三方库已经达到 9 万多个。如果说强大的标准库奠定了 Python 语言发展的基石，丰富的第三方库则是 Python 不断发展的保证。随着 Python 语言的发展，一些稳定的第三方库不断被加入标准库。

思考与练习

6.15 如何在程序中引用 jieba 库？

6.16 使用 jieba.cut()对“中华人民共和国是一个伟大的国家”进行分词，输出结果，并将该迭代器转换为列表类型。

6.17 向分词词典中加入一些新的网络用语，并编写例子观察分词效果。

程序练习 6-3：十分钟学 jieba 库



6.6 实例 10：文本词频统计

在很多情况下，会遇到这样的问题：对于一篇给定文章，希望统计其中多次出现的词语，进而概要分析文章的内容。在对网络信息进行自动检索和归档时，也会遇到同样的问题。这就是“词频统计”问题。

从思路上看，词频统计只是累加问题，即对文档中每个词设计一个计数器，词语每出现一次，相关计数器加 1。如果以词语为键，计数器为值，构成<单词>:<出现次数>的键值对，将很好地解决该问题。这就是字典类型的优势。

下面，采用字典来解决词频统计问题。该问题的 IPO 描述如下。

输入：从文件中读取一篇文章

处理：采用字典数据结构统计词语出现频率

输出：文章中最常出现的 10 个单词及出现次数

英文文本以空格或标点符号来分隔词语，获得单词并统计数量相对容易，6.6.1 节介绍统计英文文本词频的方法。中文字符之间没有天然的分隔符，需要对中文文本进行分词，6.6.2 节介绍统计中文文本词频方法。

6.6.1 Hamlet 英文词频统计

Hamlet，《哈姆雷特》，是莎士比亚的一部经典悲剧作品，讲述了克劳狄斯叔叔

谋害哈姆雷特父亲并篡取王位、哈姆雷特流浪在外并向叔叔复仇的故事。《哈姆雷特》也叫《王子复仇记》，代表着整个西方文艺复兴时期文学的最高成就，很多国内外电影都以这个故事为原型。

程序素材 6-1：哈姆雷特文本

获取该故事的文本文件，保存为 hamlet.txt。全文可以从网络上找到，或从本书提供的电子资源中获取。

统计 Hamlet 英文词频的第一步是分解并提取英文文章的单词。同一个单词会存在大小写不同形式，但计数却不能区分大小写。假设 *Hamlet* 文本由变量 txt 表示，可以通过 txt.lower()函数将字母变成小写，排除原文大小写差异对词频统计的干扰。英文单词的分隔可以是空格、标点符号或者特殊符号。为统一分隔方式，可以将各种特殊字符和标点符号使用 txt. replace()方法替换成空格，再提取单词。

统计词频的第二步是对每个单词进行计数。假设将单词保存在变量 word 中，使用一个字典类型 counts={}，统计单词出现的次数可采用如下代码：

```
counts[word] = counts[word] + 1
```

当遇到一个新词时，单词没有出现在字典结构中，则需要在字典中新建键值对：

```
counts[new_word] = 1
```

因此，无论词是否在字典中，加入字典 counts 中的处理逻辑可以统一表示如下：

```
if word in counts:
    counts[word] = counts[word] + 1
else:
    counts[word] = 1
```

或者，这个处理逻辑可以更简洁地表示为如下代码：

```
counts[word] = counts.get(word,0) + 1
```

字典类型的 counts.get(word,0)方法表示：如果 word 在 counts 中，则返回 word 对应的值，如果 word 不在 counts 中，则返回 0。

该实例的第三步是对单词的统计值从高到低进行排序，输出前 20 个高频词语，并格式化打印输出。由于字典类型没有顺序，需要将其转换为有顺序的列表类型，再使用 sort()方法和 lambda 函数配合实现根据单词出现的次数对元素进行排序。最后输出排序结果前 10 位的单词。

```
items = list(counts.items())    #将字典转换为记录列表
items.sort(key=lambda x:x[1], reverse=True)#以记录第 2 列排序
```

源代码 6-2：哈姆雷特英文词频统计



采用函数对获取和整理文本进行封装，下面给出该实例的完整代码。

实例代码 10.1　　　　e10.1CalHamlet.py

```
1  #e10.1CalHamlet.py
2  def getText():
3      txt = open("hamlet.txt", "r").read()
4      txt = txt.lower()
5      for ch in '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~':
6          txt = txt.replace(ch, " ")   #将文本中特殊字符替换为空格
7      return txt
8  hamletTxt = getText()
```

```
 9  words  = hamletTxt.split()
10  counts = {}
11  for word in words:
12     counts[word] = counts.get(word,0) + 1
13  items = list(counts.items())
14  items.sort(key=lambda x:x[1], reverse=True)
15  for i in range(10):
16     word, count = items[i]
17     print ("{0:<10}{1:>5}".format(word, count))
```

运行程序后，输出结果如下：

```
>>>
the        1138
and         965
to          754
of          669
you         550
a           542
i           542
my          514
hamlet      462
in          436
```

观察输出结果可以看到，高频单词大多数是冠词、代词、连接词等语法型词汇，并不能代表文章的含义。进一步地，可以采用集合类型构建一个排除词汇库excludes，在输出结果中排除这个词汇库中的内容，具备这样功能程序的完整代码如下：

源代码 6-3：带排除的哈姆雷特英文词频统计



实例代码 10.2　　　　e10.2CalHamlet.py

```
 1  #e10.2CalHamlet.py
 2  excludes = {"the","and","of","you","a","i","my","in"}
 3  def getText():
 4     txt = open("hamlet.txt", "r").read()
 5     txt = txt.lower()
 6     for ch in '!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~':
 7        txt = txt.replace(ch, " ")   #将文本中特殊字符替换为空格
 8     return txt
 9  hamletTxt = getText()
10  words  = hamletTxt.split()
11  counts = {}
12  for word in words:
13     counts[word] = counts.get(word,0) + 1
14  for word in excludes:
```

```
15        del(counts[word])
16    items = list(counts.items())
17    items.sort(key=lambda x:x[1], reverse=True)
18    for i in range(10):
19        word, count = items[i]
20        print ("{0:<10}{1:>5}".format(word, count))
```

运行程序后，输出结果如下：

```
>>>
to          754
hamlet      462
it          416
that        391
is          340
not         314
lord        309
his         296
this        295
but         269
```

再次输出仍然发现了很多语法型词汇，如果希望排除更多的词汇，可以继续增加 excludes 中的内容，请感兴趣的读者逐步完善这个程序功能。

6.6.2 《三国演义》人物出场统计

程序素材 6-2：三国演义文本

《三国演义》是中国古典四大名著之一，作者是元末明初的小说家罗贯中。该书描写了从东汉末年到西晋初年之间近 105 年的历史风云，以描写战争为主，反映了东汉末年的群雄割据混战和魏、蜀、吴三国之间的政治和军事斗争。

《三国演义》是一本鸿篇巨著，里面出现了几百个各具特色的人物。每次读这本经典作品都会想一个问题，全书这些人物谁出场最多呢？一起来用 Python 回答这个问题吧。

源代码 6-4：三国演义中文词频统计



人物出场统计涉及对词汇的统计。中文文章需要分词才能进行词频统计，这需要用到 jieba 库。分词后的词频统计方法与 *Hamlet* 的英文词频统计方法类似。《三国演义》文本保存为三国演义.txt。实现代码如下：

实例代码 10.3　　　　e10.3CalThreeKingdoms.py

```
1    #e10.3CalThreeKingdoms.py
2    import jieba
3    txt = open("三国演义.txt", "r", encoding='utf-8').read()
4    words  = jieba.lcut(txt)
5    counts = {}
```

```
 6  for word in words:
 7      if len(word) == 1:  #排除单个字符的分词结果
 8          continue
 9      else:
10          counts[word] = counts.get(word,0) + 1
11  items = list(counts.items())
12  items.sort(key=lambda x:x[1], reverse=True)
13  for i in range(15):
14      word, count = items[i]
15      print ("{0:<10}{1:>5}".format(word, count))
```

先输出排序前 15 的单词，运行程序后，输出结果如下：

```
>>>
曹操        953
孔明        836
将军        772
却说        656
玄德        585
关公        510
丞相        491
二人        469
不可        440
荆州        425
玄德曰      390
孔明曰      390
不能        384
如此        378
张飞        358
```

观察输出结果，似乎“曹操”是出场次数最多的人。然而，结果中出现了“玄德”、“玄德曰”，读者应该知道“玄德”就是“刘备”。同一个人物会有不同的名字，这种情况需要整合处理。同时，与英文词频统计类似，需要排除一些人名无关的词汇，如“却说”、“将军”等。进一步完善代码如下，其中，第 3 行增加了排除词库 excludes，第 10 到第 17 行增加了同一人物不同名字的处理。

源代码 6-5：三国演义中文人名词频统计



实例代码 10.4　　e10.4CalThreeKingdoms.py

```
1  #e10.4CalThreeKingdoms.py
2  import jieba
3  excludes = {"将军","却说","荆州","二人","不可","不能","如此"}
4  txt = open("三国演义.txt", "r", encoding='utf-8').read()
5  words  = jieba.lcut(txt)
6  counts = {}
7  for word in words:
```

```
8      if len(word) == 1:
9          continue
10     elif word == "诸葛亮" or word == "孔明曰":
11         rword = "孔明"
12     elif word == "关公" or word == "云长":
13         rword = "关羽"
14     elif word == "玄德" or word == "玄德曰":
15         rword = "刘备"
16     elif word == "孟德" or word == "丞相":
17         rword = "曹操"
18     else:
19         rword = word
20     counts[rword] = counts.get(rword,0) + 1
21 for word in excludes:
22     del(counts[word])
23 items = list(counts.items())
24 items.sort(key=lambda x:x[1], reverse=True)
25 for i in range(5):
26     word, count = items[i]
27     print ("{0:<10}{1:>5}".format(word, count))
```

输出排序前 5 的单词，运行程序后，输出结果如下：

```
>>>
曹操        1451
孔明        1383
刘备        1252
关羽         784
张飞         358
```

由此可以获得结论，“曹操”、“孔明”和“刘备”是《三国演义》中出场次数最多的人，他们之间的出场次数不相上下，随后是“关羽”和“张飞”。请感兴趣的读者继续完善程序，排除更多无关词汇干扰，总结出场最多的 20 个人物。这里给出参考答案。

曹操（1451）、孔明（1383）、刘备（1252）、关羽（784）、张飞（358）、
吕布（300）、赵云（278）、孙权（264）、司马懿（221）、周瑜（217）、
袁绍（191）、马超（185）、魏延（180）、黄忠（168）、姜维（151）、
马岱（127）、庞德（122）、孟获（122）、刘表（120）、夏侯惇（116）

拓展： 自然语言处理

自然语言处理是计算机科学领域与人工智能领域相结合的一个重要方向。它研究人与计算机之间用自然语言进行有效通信的理论和方法。自然语言处理是一门融语言学、计算机科学、数学于一体的科学。自然语言的研究有地域性，少数民族语言、中文和各种外国语言都有独特的自然语言处理方法。

思考与练习

6.18　理解实例代码 10.1 列表排序中调用 lambda 函数的使用。

6.19　程序中哪段代码实现了将字典转换为列表？

6.7　实例 11：Python 之禅

源代码 6-6：Python 之禅



要点： Python 之禅介绍了编写好代码的基本原则。

什么样的程序是好的？如何编写漂亮的代码？这是学习编程一段时间最经常提出的问题，却最难回答。程序设计语言如同自然语言一样，好的代码就像文学作品，不仅达意，更要优美。那什么是"好"？什么是"优美"？领悟编程代码优美的过程类似参禅，除了不断练习，也需要理解一些原则。

Python 编译器以函数库的形式内置了一个有趣的文件，被称为"Python 之禅"（The Zen of Python）。当调用如下一行语句后，会出现一段有趣的运行结果。

```
>>>import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

这是一篇由 Tim Peters 撰写的文章，介绍了编写优美的 Python 程序所需要关注的一些重要原则。这里给出了作者对 Python 之禅的一个参考翻译及一些编程体会，

供读者参考。

Python 之禅　作者：Tim Peters	译者心得
• 优美胜于丑陋	以编写优美代码为目标，不多解释
• 明了胜于隐晦	优美代码应该清晰明了，规范统一
• 简洁胜于复杂	优美代码应该逻辑简洁，避免复杂逻辑
• 复杂胜于凌乱	如果必须采用复杂逻辑，接口关系也要清晰
• 扁平胜于嵌套	优美代码应该是扁平的，避免太多层次嵌套
• 间隔胜于紧凑	优美代码间隔要适当，每行代码解决适度问题
• 可读性很重要	优美代码必须是可读且易读的
• 即便假借特例的实用性之名，也不要违背上述规则	上述规则是至高无上的
• 除非你确定需要，任何错误都应该有应对	捕获异常，不让程序留有因错误退出的可能
• 当存在多种可能，不要尝试去猜测	
• 只要你不是 Guido，对于问题尽量找一种，最好是唯一明显的解决方案	不要试图给出多种方案，找到一种实现它，几乎所有人都没有 Guido 那么牛
• 做也许好过不做，但不假思索就动手还不如不做	编程之前要有思考
• 如果你无法向人描述你的实现方案，那肯定不是一个好方案	能说清楚的往往才是对的
• 如果实现方案容易解释，可能是个好方案	
• 命名空间是绝妙的理念，要多运用	适合复杂程序编程

拓展： 代码的艺术

好的程序代码不仅能够完成功能，也是一件艺术作品。对于程序员来说，绝大多数时间在阅读并修改代码，好的代码可读性能将这种阅读变成一种享受。如何能够编写好的代码呢？请大家首先熟练掌握一门语言，如 Python；其次，要阅读一定量优质的代码，思考这些代码的编程思路和风格；最后，选择一个有兴趣的项目编写代码、修改、再修改，直到你认为它是个作品为止。

除了 Python 之禅所表达的 Python 设计理念，该程序还有另一段魅力。请读者在 Python 安装目录中找到 Lib/this.py 文件，该程序内容如下：

源代码 6-7：Python 之禅源代码



实例代码 11.1　　　　this.py

```
1  s = """Gur Mra bs Clguba, ol Gvz Crgref
2
3  Ornhgvshy vf orggre guna htyl.
4  Rkcyvpvg vf orggre guna vzcyvpvg.
5  Fvzcyr vf orggre guna pbzcyrk.
6  Pbzcyrk vf orggre guna pbzcyvpngrq.
7  Syng vf orggre guna arfgrq.
```

```
8   Fcnefr vf orggre guna qrafr.
9   Ernqnovyvgl pbhagf.
10  Fcrpvny pnfrf nera'g fcrpvny rabhtu gb oernx gur ehyrf.
11  Nygubhtu cenpgvpnyvgl orngf chevgl.
12  Reebef fubhyq arire cnff fvyragyl.
13  Hayrff rkcyvpvgyl fvyraprq.
14  Va gur snpr bs nzovthvgl, ershfr gur grzcgngvba gb thrff.
15  Gurer fubhyq or bar-- naq cersrenoyl bayl bar --boivbhf jnl gb
16  qb vg.
17  Nygubhtu gung jnl znl abg or boivbhf ng svefg hayrff lbh'er Qhgpu.
18  Abj vf orggre guna arire.
19  Nygubhtu arire vf bsgra orggre guna *evtug* abj.
20  Vs gur vzcyrzragngvba vf uneq gb rkcynva, vg'f n onq vqrn.
21  Vs gur vzcyrzragngvba vf rnfl gb rkcynva, vg znl or n tbbq vqrn.
22  Anzrfcnprf ner bar ubaxvat terng vqrn -- yrg'f qb zber bs gubfr!"""
23
24  d = {}
25  for c in (65, 97):
26      for i in range(26):
27          d[chr(i+c)] = chr((i+13) % 26 + c)
28  print("".join([d.get(c, c) for c in s]))
```

该程序第 1 到第 23 行是一个字符串 s，但该字符串并非明文。将 s 转换成明文内容的代码是从第 25 行到第 30 行。在这些代码中，可以看到字典、列表和元组类型，下面逐行来阅读这个代码的功能。

第 24 行定义了一个空字典 d，第 27 行对字典 d 填充了内容，这个填充将 i+c 对应的字符替换为(i+13)%26 +c，即将编号循环增加了 13。chr(65)代表字符'A'，chr(97)代表字符'a'，因此，第 24 到第 27 行建立了字母 a 到 z 和字母 A 到 Z 的一个 13 位循环移动的对应表，如下所示：

```
密文: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
原文: N O P Q R S T U V W X Y Z A B C D E F G H I J K L M
密文: a b c d e f g h i j k l m n o p q r s t u v w x y z
原文: n o p q r s t u v w x y z a b c d e f g h i j k l m
```

这个算法可以看作是 3.5 节介绍的凯撒密码的一种扩展，相比凯撒密码，这个算法采用循环移动 13 个位置，直接好处是原文和密文之间的相互转换可以使用同一个程序，建议读者掌握这个算法，传递个小纸条、发个小消息就不怕被别人看懂了。

思考与练习

6.20　请列出 5 条写出优美代码的编程原则。

6.21　实例代码 11.1 中第 27 行的功能是什么？

本 章 小 结

本章主要介绍了组合数据类型中元组、数组、列表和字典等类型及基本操作，讲解了如何使用 jieba 词库对中文文档进行分词并进一步统计文档词频，最后利用 Python 之禅的例子介绍了编写好代码的基本原则。

程序练习题

程序练习 6-4：章节程序练习题



6.1 随机密码生成。编写程序，在 26 个字母大小写和 9 个数字组成的列表中随机生成 10 个 8 位密码。

6.2 重复元素判定。编写一个函数，接受列表作为参数，如果一个元素在列表中出现了不止一次，则返回 True，但不要改变原来列表的值。同时编写调用这个函数和测试结果的程序。

6.3 重复元素判定续。利用集合的无重复性改编程序练习题 6.2 的程序，获得一个更快更简洁的版本。

6.4 文本字符分析。编写程序接收字符串，按字符出现频率的降序打印字母。分别尝试录入一些中英文文章片段，比较不同语言之间字符频率的差别。

6.5 生日悖论分析。生日悖论指如果一个房间里有 23 人或以上，那么至少有两个人生日相同的概率大于 50%。编写程序，输出在不同随机样本数量下，23 个人中至少两个人生日相同的概率。

6.6 《红楼梦》人物统计。编写程序统计《红楼梦》中前 20 位出场最多的人物。

第 7 章　文件和数据格式化

电子教案 7-1：文件和数据格式化



最有效的调试工具是静下心来仔细思考，辅之审慎放置的打印语句。

The most effective debugging tool is still careful thought, coupled with judiciously placed print statements.

——布赖恩・克尼汉（Brian W. Kernighan）

著名的计算机科学家，参加了 UNIX 系统和 C 语言的开发

计算机程序设计领域许多具有影响力著作的作者

学习目标

（1）掌握文件的读写方法以及打开和关闭等基本操作。
（2）理解数据组织的维度及其特点。
（3）掌握一二维数据的存储格式和读写方法。
（4）运用 PIL 库进行基本的图像处理。
（5）运用 json 库进行数据的维度转换。
（6）了解高维数据的存储格式和读写方法。

字符画(ASCII Art)的历史可以追溯到计算机诞生之初，起初用在图形显示功能受限的设备上，用 ASCII 字符集中的可打印字符拼成图片。如今，高清显示屏早已普及，字符画更多被当作一门艺术来欣赏，很多人喜欢用字符画处理自己的照片作为独一无二的创意头像。

想了解如何用不超过 50 行 Python 代码生成一幅精美的字符画吗？

7.1 文件的使用

> 要点：Python 能够以文本和二进制两种方式处理文件。

7.1.1 文件概述

文件是一个存储在辅助存储器上的数据序列，可以包含任何数据内容。概念上，文件是数据的集合和抽象，类似地，函数是程序的集合和抽象。用文件形式组织和表达数据更有效也更为灵活。文件包括两种类型：文本文件和二进制文件。

文本文件一般由单一特定编码的字符组成，如 UTF-8 编码，内容容易统一展示和阅读。大部分文本文件都可以通过文本编辑软件或文字处理软件创建、修改和阅读。由于文本文件存在编码，因此，它也可以被看作是存储在磁盘上的长字符串，例如一个 txt 格式的文本文件。

二进制文件直接由比特 0 和比特 1 组成，没有统一字符编码，文件内部数据的组织格式与文件用途有关。二进制是信息按照非字符但特定格式形成的文件，例如，png 格式的图片文件、avi 格式的视频文件。二进制文件和文本文件最主要的区别在于是否有统一的字符编码。二进制文件由于没有统一字符编码，只能当作字节流，而不能看作是字符串。

无论文件是创建为文本文件还是二进制文件，都可以用“文本文件方式”和“二进制文件方式”打开，但打开后的操作不同。

源代码 7-1：理解文本文件和二进制文件的区别



【微实例 7.1】理解文本文件和二进制文件的区别。

首先，用文本编辑器生成一个包含“中国是个伟大的国家！”的 txt 格式文本文件，命名为 7.1.txt。分别用文本文件方式和二进制文件方式读入，并打印输出效果，代码如下：

微实例 7.1　　m7.1DiffTextBin.py

```
1  textFile = open("7.1.txt","rt") #t 表示文本文件方式
2  print(textFile.readline())
3  textFile.close()
4  binFile = open("7-1.txt","rb")   #r 表示二进制文件方式
5  print(binFile.readline())
6  binFile.close()
```

输出结果如下：

```
>>>
中国是个伟大的国家！
```

```
b'\xd6\xd0\xb9\xfa\xca\xc7\xb8\xf6\xce\xb0\xb4\xf3\xb5\xc4\xb9\xfa\xb
c\xd2\xa3\xa1'
```

可以看到，采用文本方式读入文件，文件经过编码形成字符串，打印出有含义的字符；采用二进制方式打开文件，文件被解析为字节（Byte）流。由于存在编码，字符串中的一个字符由两个字节表示。

7.1.2　文件的打开关闭

Python 对文本文件和二进制文件采用统一的操作步骤，即“打开—操作—关闭”，如图 7.1 所示。操作系统中的文件默认处于存储状态，首先需要将其打开，使得当前程序有权操作这个文件，打开不存在的文件可以创建文件。打开后的文件处于占用状态，此时，另一个进程不能操作这个文件。可以通过一组方法读取文件的内容或向文件写入内容，此时，文件作为一个数据对象存在，采用<a>.<b>()方式进行操作。操作之后需要将文件关闭，关闭将释放对文件的控制使文件恢复存储状态，此时，另一个进程将能够操作这个文件。



图 7.1　文件的状态和操作过程

Python 通过解释器内置的 open()函数打开一个文件，并实现该文件与一个程序变量的关联，open()函数格式如下：

```
<变量名> = open(<文件名>, <打开模式>)
```

open()函数有两个参数：文件名和打开模式。文件名可以是文件的实际名字，也可以是包含完整路径的名字。打开模式用于控制使用何种方式打开文件，open()函数提供 7 种基本的打开模式，如表 7.1 所示。

表 7.1 文件的打开模式（共 7 个）

文件的打开模式	含　义
'r'	只读模式，如果文件不存在，返回异常 FileNotFoundError，默认值
'w'	覆盖写模式，文件不存在则创建，存在则完全覆盖
'x'	创建写模式，文件不存在则创建，存在则返回异常 FileExistsError
'a'	追加写模式，文件不存在则创建，存在则在文件最后追加内容
'b'	二进制文件模式
't'	文本文件模式，默认值
'+'	与 r/w/x/a 一同使用，在原功能基础上增加同时读写功能

打开模式使用字符串方式表示，根据字符串定义，单引号或者双引号均可。上述打开模式中，'r'、'w'、'x'、'b'可以和'b'、't'、'+'组合使用，形成既表达读写又表达文件模式的方式。例如，open()函数默认采用'rt'（文本只读）模式，读入程序所在目录中 7.1.txt 文件：

```
textfile = open('7.1.txt', 'r')
```

或

```
textfile = open('7.1.txt')
```

读取一个二进制文件，如一张图片、一段视频或者一段音乐，需要使用文件打开模式'rb'。例如，打开一个名为“music.mp3”的音频文件：

```
binfile = open('music.mp3', 'rb')
```

文件使用结束后要用 close()方法关闭，释放文件的使用授权，该方法的使用方式如下：

```
<变量名>.close()
```

7.1.3 文件的读写

当文件被打开后，根据打开方式不同可以对文件进行相应的读写操作。注意，当文件以文本文件方式打开时，读写按照字符串方式，采用当前计算机使用的编码或指定编码；当文件以二进制文件方式打开时，读写按照字节流方式。 Python 提供 4 个常用的文件内容读取方法，如表 7.2 所示。

表 7.2 文件内容读取方法（共 4 个）

操作方法	含 义
<file>.readall()	读入整个文件内容，返回一个字符串或字节流*
<file>.read(size=-1)	从文件中读入整个文件内容，如果给出参数，读入前 size 长度的字符串或字节流
<file>.readline(size = -1)	从文件中读入一行内容，如果给出参数，读入该行前 size 长度的字符串或字节流
<file>.readlines(hint=-1)	从文件中读入所有行，以每行为元素形成一个列表，如果给出参数，读入 hint 行

*：字符串或字节流取决于文件打开模式，如果是文本方式打开，返回字符串；否则返回字节流。下同。

源代码 7-2：文本文件逐行打印



【微实例 7.2】文本文件逐行打印。

用户输入文件路径，以文本文件方式读入文件内容并逐行打印，代码如下：

微实例 7.2　　m7.2PrintFilebyLines.py

```
1  fname = input("请输入要打开的文件： ")
2  fo = open(fname, "r")
3  for line in fo.readlines():
```

```
4       print(line)
5   fo.close()
```

程序首先提示用户输入一个文件名，然后打开文件并赋值给文件对象变量 fo。文件的全部内容通过 fo.readlines()方法读入到一个列表中，列表的每个元素是文件一行的内容，然后通过 for-in 方式遍历列表，处理每行内容。

上述代码尽管完成了微实例 7.2 的要求，但存在一些缺点：当读入文件非常大时，一次性将内容读取到列表中会占用很多内存，影响程序执行速度。一个合理的方法是逐行读入内容到内存，并逐行处理。这可以通过一个简单的方法解决。Python 将文件本身作为一个行序列，遍历文件的所有行可以直接这样完成：

```
1   fname = input("请输入要打开的文件：")
2   fo = open(fname, "r")
3   for line in fo:
4       print(line)
5   fo.close()
```

如果程序需要逐行处理文件内容，建议采用上述代码中第 2～5 行组成的格式，如下：

```
fo = open(fname, "r")
for line in fo:
    # 处理一行数据
fo.close()
```

拓展： 文件的换行符

如果采用二进制方式打开文件，换行符只是一个符号，对应一个字节，表示为"\n"；如果采用文本方式打开文件，换行符表示一行的结束，辅助程序对文件的处理。文件的换行符是真实存在的一个字符。

Python 提供 3 个与文件内容写入有关的方法，如表 7.3 所示。

表 7.3 文件内容写入方法（共 3 个）

方　法	含　义
<file>.write(s)	向文件写入一个字符串或字节流
<file>.writelines(lines)	将一个元素全为字符串的列表写入文件
<file>.seek(offset)	改变当前文件操作指针的位置，offset 的值： 0——文件开头；1——当前位置；2——文件结尾

【微实例 7.3】向文件写入一个列表。

向文件写一个列表类型，并打印输出结果，代码如下：

源代码 7-3：向文件写入一个列表

微实例 7.3　　m7.3WriteListtoFile.py

```
1  fname = input("请输入要写入的文件: ")
2  fo = open(fname, "w+")
3  ls = ["唐诗", "宋词", "元曲"]
4  fo.writelines(ls)
5  for line in fo:
6      print(line)
7  fo.close()
```

程序执行结果如下：

```
>>>请输入要写入的文件: test.txt
>>>
```

可以看到，程序并没有输出写入的列表内容。在 Write Listto File.py 程序的目录中找到 test.txt 文件，打开可以看到其中的内容如下：

```
唐诗宋词元曲
```

列表 ls 内容被写入文件，但为何第 5 至第 7 行代码没有将这些内容打印出来呢？这是因为文件写入内容后，当前文件操作指针在写入内容的后面，第 5 至第 7 行代码从指针开始向后读入并打印内容，被写入的内容却在指针前面，因此未能被打印出来。为此，可以在写入文件后增加一条代码 fo.seek(0)将文件操作指针返回到文件开始，即可显示写入的内容，代码如下：

```
1  fname = input("请输入要写入的文件: ")
2  fo = open(fname, "w+")
3  ls = ["唐诗", "宋词", "元曲"]
4  fo.writelines(ls)
5  fo.seek(0)
6  for line in fo:
7      print(line)
8  fo.close()
```

程序执行结果如下：

```
>>>请输入要写入的文件: test.txt
唐诗宋词元曲
```

可能会有读者认为 fo.writelines(ls)写入后的内容与预期不符，因为 writelines()正如其名，该函数似乎应该把每个元素写入文件的单独一行，即每个列表元素写入内容后应该有换行，但实际上却没有。fo.writelines()方法并不在列表后面增加换行，只是将列表内容直接排列输出。

阶段测试 7-1：Python 文件使用小测验



思考与练习

7.1　读写文件需要采用 open()函数打开文件，采用绝对路径打开操作系统中的一个文件。

7.2　若文件不存在，采用读取方式时，会发生什么情况？采用写入方式时，又会发生什么情况？

7.3　如何从文件中读取 30 个字符？

7.4　下列不是 Python 对文件的读操作方法的是（　　）

A．read　　B．readline　　C．readall　　D．readtext

7.5　采用 open()函数打开 Windows 系统目录（C:\Windows 或其他系统安装目录）中的一个文件，会出现什么情况？

程序练习 7-1：半小时学 Python 文件



7.2　模块 5：PIL 库的使用

> **要点：** PIL 库是一个具有强大图像处理能力的第三方库，不仅包含了丰富的像素、色彩操作功能，还可以用于图像归档和批量处理。

7.2.1　PIL 库概述

PIL（Python Image Library）库是 Python 语言的第三方库，需要通过 pip 工具安装，Python 安装第三方库的详细方法请见 8.6 节。安装 PIL 库的方法如下，需要注意，安装库的名字是 pillow。

```
:\>pip install pillow  # 或者 pip3 install pillow
```

图片资料 7-1：Python 快速参考之 PIL 库



PIL 库支持图像存储、显示和处理，它能够处理几乎所有图片格式，可以完成对图像的缩放、剪裁、叠加以及向图像添加线条、图像和文字等操作。

PIL 库主要可以实现图像归档和图像处理两方面功能需求。

（1）图像归档：对图像进行批处理、生成图像预览、图像格式转换等。

（2）图像处理：图像基本处理、像素处理、颜色处理等。

根据功能不同，PIL 库共包括 21 个与图片相关的类，这些类可以被看作是子库或 PIL 库中的模块，子库列表如下。

Image、ImageChops、ImageColor、ImageCrackCode、ImageDraw、ImageEnhance、ImageFile、ImageFileIO、ImageFilter、ImageFont、ImageGL、ImageGrab、Imagemath、ImageOps、ImagePalette、ImagePath、ImageQt、ImageSequence、ImageStat、ImageTk、ImageWin

本书重点介绍 PIL 库最常用的 4 个子库：Image、ImageFilter、ImageEnhance。更多 PIL 库内容请参考网站：http://effbot.org/imagingbook/。

7.2.2 PIL 库 Image 类解析

Image 是 PIL 最重要的类，它代表一张图片，引入这个类的方法如下：

```
>>>from PIL import Image
```

在 PIL 中，任何一个图像文件都可以用 Image 对象表示。表 7.4 给出了 Image 类的图像读取和创建方法。

表 7.4 Image 类的图像读取和创建方法（共 5 个）

方 法	描 述
Image.open(filename)	根据参数加载图像文件
Image.new(mode, size, color)	根据给定参数创建一个新的图像
Image.open(StringIO.StringIO(buffer))	从字符串中获取图像
Image.frombytes(mode, size, data)	根据像素点 data 创建图像
Image.verify()	对图像文件完整性进行检查，返回异常

通过 Image 打开图像文件时，图像的栅格数据不会被直接解码或者加载，程序只是读取了图像文件头部的元数据信息，这部分信息标识了图像的格式、颜色、大小等。因此，打开一个文件会十分迅速，与图像的存储和压缩方式无关。

要加载一个图像文件，最简单的形式如下，之后所有操作对 im 起作用。

```
>>>from PIL import Image
>>>im = Image.open("D:\\pycodes\\birdnest.jpg")
```

其中，birdnest.jpg 是一张鸟巢的夜景图像，存储在 D:\pycodes 目录中，如图 7.2 所示。在使用 IDLE 交互方式处理图片文件时，建议采用文件的全路径；如果使用 Python 文件形式，建议采用相对路径，将文件和程序放到一个目录中，例如：

```
1  from PIL import Image
2  im = Image.open("birdnest.jpg")
```

Image 类有 4 个处理图片的常用属性，如表 7.5 所示。

表 7.5 Image 类的常用属性（共 4 个）

属 性	描 述
Image.format	标识图像格式或来源，如果图像不是从文件读取，值为 None
Image.mode	图像的色彩模式，"L"为灰度图像、"RGB"为真彩色图像、"CMYK"为出版图像
Image.size	图像宽度和高度，单位是像素（px），返回值是二元元组（tuple）
Image.palette	调色板属性，返回一个 ImagePalette 类型

查看已经读取的图像文件的属性如下：

```
>>>print(im.format,im.size,im.mode)
JPEG (900, 598) RGB
```

拓展： CMYK 色彩

CMYK 色彩是彩色印刷时采用的一套色彩体系，也称印刷四色。CMYK 利用色料的三原色和黑色墨油混合叠加，形成各种色彩。与 RGB 用于电子显示的颜色不同，印刷中颜色使用油料叠加，混色原理不同。其中，C 是青色、M 是品红色、Y 是黄色、K 是定位套板色（黑色）。

Image 还能读取序列类图像文件，包括 GIF、FLI、FLC、TIFF 等格式文件。open()方法打开一个图像时自动加载序列中的第一帧，使用 seek()和 tell()方法可以在不同帧之间移动，如表 7.6 所示。

表 7.6　Image 类的序列图像操作方法（共 2 个）

方　法	描　述
Image.seek(frame)	跳转并返回图像中的指定帧
Image.tell()	返回当前帧的序号

源代码 7-4：
GIF 文件图像提取



【微实例 7.4】GIF 文件图像提取。

对一个 GIF 格式动态文件，提取其中各帧图像，并保存为文件。

微实例 7.4　　m7.4　GifExtractor.py

```
1  from PIL import Image
2  im = Image.open('pybit.gif')        # 读入一个 GIF 文件
3  try:
4      im.save('picframe{:02d}.png'.format(im.tell()))
5      while True:
6          im.seek(im.tell()+1)
7          im.save('picframe{:02d}.png'.format(im.tell()))
8  except:
9      print("处理结束")
```

微实例 7.4 展示了一种采用 try-except 编写程序的方法，通过 seek()方法和 save()方法配合提取 GIF 图像格式的每一帧，并保存为文件。

Image 类的图像转换和保存方法如表 7.7 所示。

表 7.7　Image 类的图像转换和保存方法（共 3 个）

方　法	描　述
Image.save(filename, format)	将图像保存为 filename 文件名，format 是图片格式
Image.convert(mode)	使用不同的参数，转换图像为新的模式
Image.thumbnail(size)	创建图像的缩略图，size 是缩略图尺寸的二元元组

其中，save()方法有两个参数：文件名 filename 和图像格式 format。如果调用时不指定保存格式，如微实例 7.4，PIL 将自动根据文件名 filename 后缀存储图像；如果指定格式，则按照格式存储。搭配采用 open()和 save()方法可以实现图像的格式转换，例如，将 png 格式转换为 jpg 格式，代码如下。需要注意，Image 类的 save()方法主要用于保存文件到硬盘，PIL 库还提供了功能更强大的格式转换方法。

```
1  im = Image.open("birdnest.jpg")
2  im.save("birdnest.png")
```

生成"birdnest.jpg"图像的缩略图，代码如下（续上一个 IDLE 指令），鸟巢图片及其缩略图如图 7.2 所示，其中（128,128）是缩略图的尺寸。

```
>>>im.thumbnail((128, 128))
>>>im.save("birdnestTN","JPEG")
```

彩图素材 7-1：北京鸟巢图片





(a) 北京鸟巢图片



(b) 缩略图

图 7.2 北京鸟巢图片及其缩略图

Image 类可以缩放和旋转图像，方法如表 7.8 所示，其中，rotate()方法以逆时针旋转的角度值作为参数来旋转图像。

表 7.8 Image 类的图像旋转和缩放方法（共 2 个）

方　法	描　述
Image.resize(size)	按 size 大小调整图像，生成副本
Image.rotate(angle)	按 angle 角度旋转图像，生成副本

Image 类能够对每个像素点或者一幅 RGB 图像的每个通道单独进行操作，如表 7.9 所示。split()方法能够将 RGB 图像各颜色通道提取出来，merge()方法能够将各独立通道再合成一幅新的图像。

表 7.9　Image 类的图像像素和通道处理方法（共 4 个）

方　　法	描　　述
Image.point(func)	根据函数 func 的功能对每个元素进行运算，返回图像副本
Image.split()	提取 RGB 图像的每个颜色通道，返回图像副本
Image.merge(mode,bands)	合并通道，其中 mode 表示色彩，bands 表示新的色彩通道
Image.blend(im1,im2,alpha)	将两幅图片 im1 和 im2 按照如下公式插值后生成新的图像：im1 * (1.0-alpha) + im2 * alpha

【微实例 7.5】图像的颜色交换。

交换图像中的颜色。可以通过分离 RGB 图片的 3 个颜色通道实现颜色交换。代码如下，程序执行效果如图 7.3 所示，夜色下的北京鸟巢变成了蓝色！

源代码 7-5：图像的颜色交换



微实例 7.5　　m7.5ChangeRGB.py

```
1  from PIL import Image
2  im = Image.open('birdnest.jpg')
3  r, g, b = im.split()
4  om = Image.merge("RGB", (b, g, r))
5  om.save('birdnestBGR.jpg')
```

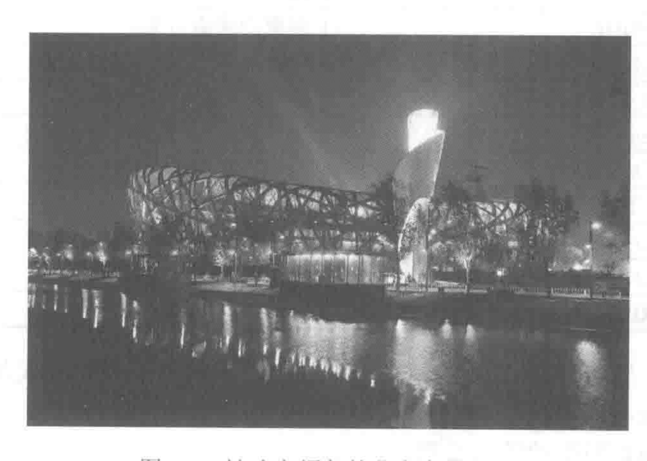


图 7.3　被改变颜色的北京鸟巢图片

彩图素材 7-2：被改变颜色的北京鸟巢图片



操作图像的每个像素点需要通过函数实现，可以采用 lambda 函数和 point()方法，例子如下，显示效果如图 7.4 所示。

```
>>>im = Image.open('D:\\pycodes\\birdnest.jpg') #打开鸟巢文件
>>>r, g, b = im.split() #获得 RGB 通道数据
>>>newg = g.point(lambda i: i * 0.9) # 将 G 通道颜色值变为原来的 0.9 倍
>>>newb = b.point(lambda i: i < 100) # 选择 B 通道值低于 100 的像素点
>>>om = Image.merge(im.mode, (r, newg, newb)) # 将 3 个通道合成为新图像
>>>om.save('D:\\pycodes\\birdnestMerge.jpg') #输出图片
```

彩图素材 7-3：
去掉光线的北京鸟巢图片



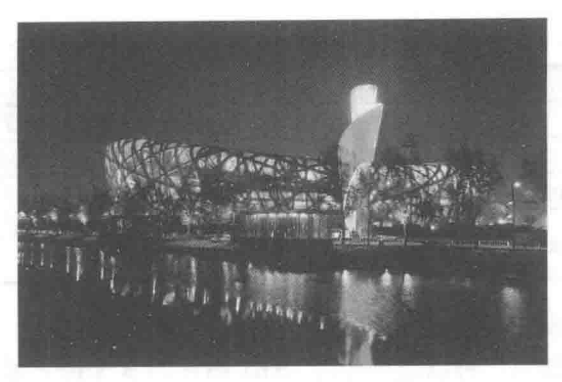


图 7.4 去掉光线的北京鸟巢图片

7.2.3 图像的过滤和增强

PIL 库的 ImageFilter 类和 ImageEnhance 类提供了过滤图像和增强图像的方法。ImageFilter 类共提供 10 种预定义图像过滤方法，如表 7.10 所示。

表 7.10 ImageFilter 类的预定义过滤方法（共 10 个）

方法表示	描述
ImageFilter.BLUR	图像的模糊效果
ImageFilter.CONTOUR	图像的轮廓效果
ImageFilter.DETAIL	图像的细节效果
ImageFilter.EDGE_ENHANCE	图像的边界加强效果
ImageFilter.EDGE_ENHANCE_MORE	图像的阈值边界加强效果
ImageFilter.EMBOSS	图像的浮雕效果
ImageFilter.FIND_EDGES	图像的边界效果
ImageFilter.SMOOTH	图像的平滑效果
ImageFilter.SMOOTH_MORE	图像的阈值平滑效果
ImageFilter.SHARPEN	图像的锐化效果

源代码 7-6：
图像的轮廓获取



利用 Image 类的 filter()方法可以使用 ImageFilter 类，使用方式如下：

```
Image.filter(ImageFilter.fuction)
```

【微实例 7.6】图像的轮廓获取。

获取图像的轮廓，代码如下，程序执行效果如图 7.5 所示，北京鸟巢变得更加抽象、更具想象空间！

微实例 7.6　　m7.6GetImageContour.py

```
1  from PIL import Image
2  from PIL import ImageFilter
3  im = Image.open('birdnest.jpg')
4  om = im.filter(ImageFilter.CONTOUR)
5  om.save('birdnestContour.jpg')
```

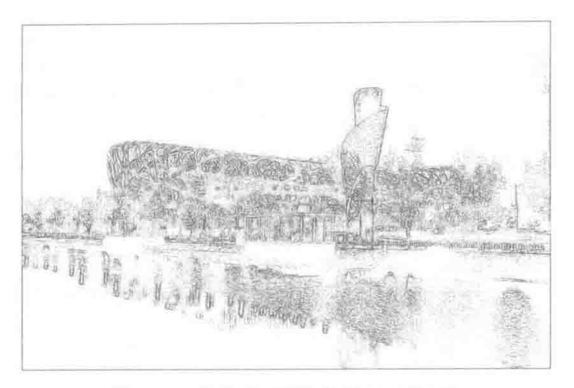


图 7.5　北京鸟巢图片的轮廓效果

彩图素材 7-4：北京鸟巢图片的轮廓效果



ImageEnhance 类提供了更高级的图像增强功能，如调整色彩度、亮度、对比度、锐化等，如表 7.11 所示。

表 7.11　ImageEnhance 类的图像增强和滤镜方法（共 5 个）

方　法	描　述
ImageEnhance.enhance(factor)	对选择属性的数值增强 factor 倍
ImageEnhance.Color(im)	调整图像的颜色平衡
ImageEnhance.Contrast(im)	调整图像的对比度
ImageEnhance.Brightness(im)	调整图像的亮度
ImageEnhance.Sharpness(im)	调整图像的锐度

【微实例 7.7】图像的对比度增强。

增强图像的对比度为初始的 20 倍。代码如下，程序执行效果如图 7.6 所示。

源代码 7-7：图像的对比度增强



微实例 7.7　　m7.7EnImageContrast.py

```
1  from PIL import Image
2  from PIL import ImageEnhance
3  im = Image.open('birdnest.jpg')
4  om = ImageEnhance.Contrast(im)
5  om.enhance(20).save('birdnestEnContrast.jpg')
```

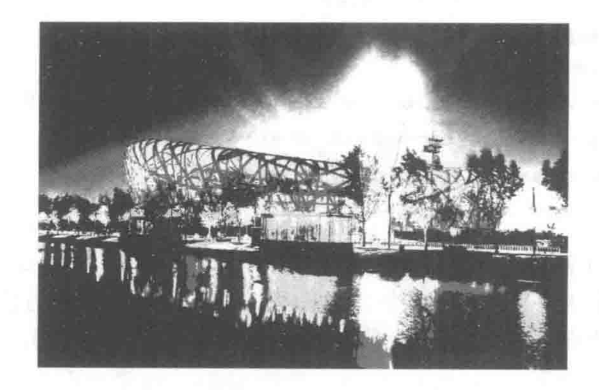


图 7.6　北京鸟巢图片的 20 倍对比度增强效果

彩图素材 7-5：北京鸟巢图片的 20 倍对比度增强效果



程序练习 7-2：
半小时学 PIL 库



思考与练习

7.6 请描述 PIL 库的 Image 类、ImageFilter 类和 ImageEnhance 类的基本功能。

7.7 如何通过 PIL 库打开并存储一个图像？

7.8 采用 PIL 库将图像中红色系去掉有哪些步骤？

7.3 实例 12：图像的字符画绘制

要点： 这是一个采用 PIL 库将图片转换为字符画的实例。

位图图片是由不同颜色像素点组成的规则分布，如果采用字符串代替像素，图像就成为了字符画。

首先自定义一个字符集，将这个字符集替代图像中的像素点，使得每个字符对应图像中的不同颜色。字符的种类越多则越能还原图像中的色彩变化，图片也更加富有层次感。

```
1  ascii_char =list("$@B%8&WM#*oahkbdpqwmZO0QLCJUYXzcvunxrjf\
2                   1234568795t/\|()1{}[]?-_+~<>i!;:,\"^`'.")
```

图像的色彩信息无法被黑白 ASCII 字符直接模拟，可以使用灰度值将彩色图像转换为高质量的黑白文稿。灰度值指黑白图像中的颜色深度，白色为 255，黑色为 0。这里定义，灰度值从大到小依次使用字符集中从左到右的符号，因此，可以直接求出不同灰度值在字符集中对应的字符编号。

定义彩色向灰度的转换公式如下，其中 R、G、B 分别是像素点的 RGB 颜色值：

```
Gray = R * 0.2126 + G * 0.7152 + B * 0.0722
```

因此，像素的 RGB 颜色值与字符集的对应函数如下：

```
1  def get_char(r, b, g, alpha=256):
2      if alpha == 0:
3          return ' '
4      gray = int(0.2126 * r + 0.7152 * g + 0.0722 * b)
5      unit = 256 / len(ascii_char)
6      return ascii_char[gray//unit]
```

为了使生成的字符画有最佳效果，可以利用 PIL 库中 Image 类的 resize(size)函数对图片重新设定大小。size 是一个二元元组，分别表示新图像的长度和宽度。resize()函数不是简单地改变图像大小，而是对像素在新尺寸下重新排列。

创建一个空字符串 txt，然后利用一个嵌套循环向里面添加字符。im.getpixel()方法可以返回给定图像位置的像素值，如果图像为多通道，则返回一个 RGB 颜色

元组。

为了使字符画更加漂亮，这里有一个小技巧。不同的字符给人带来的视觉效果是不同的，@#￥这类字符有浓密的色彩感，而_!{}/|这类字符空白较多，一般适合表示浅色。在生成字符画后，可以根据字符集和图像的对照适当修改字符排列顺序，例如，将背景色对应的字符修改为_或/，将会更加突出图像效果，将图像中浓墨重彩的地方使用@*或 B&表示会使层次感更强。

源代码 7-8：图像的字符画绘制



实例代码 12.1 如下：

实例代码 12.1　　　　e12.1DrawCharImage.py

```
1   #e12.1DrawCharImage.py
2   from PIL import Image
3   ascii_char  = list('"$%_&WM#*oahkbdpqwmZO0QLCJUYXzcvunxr\
                      jft/\|()1{}[]?-/+@<>i!;:,\^`.')
4   def get_char(r, b, g, alpha=256):
5       if alpha == 0:
6           return ' '
7       gray = int(0.2126 * r + 0.7152 * g + 0.0722 * b)
8       unit = 256 / len(ascii_char)
9       return ascii_char[int(gray//unit)]
10  def main():
11      im = Image.open('astro.jpg')
12      WIDTH, HEIGHT = 100, 60
13      im = im.resize((WIDTH, HEIGHT))
14      txt = ""
15      for i in range(HEIGHT):
16          for j in range(WIDTH):
17              txt += get_char(*im.getpixel((j, i)))
18          txt += '\n'
19      fo = open("pic_char.txt","w")
20      fo.write(txt)
21      fo.close()
22  main()
```

采用一张有十二星座的图像，命名为“astro.jpg”，如图 7.7 所示。程序执行后生成的字符画如图 7.8 所示。

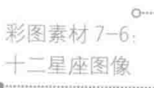






图 7.7　十二星座图像

彩图素材 7-7：十二星座图像的字符画



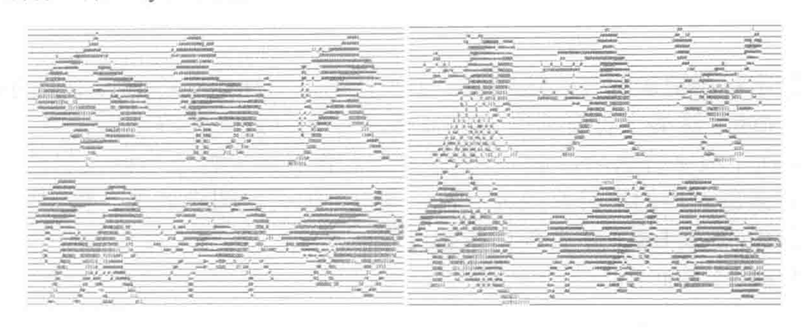


图 7.8　十二星座图像的字符画

拓展： 位图和矢量图

位图图像,亦称为点阵图像，是最为常用的图像种类，它由像素阵列组成。每个像素点颜色不同，放大位图可以看到构成图像的基本像素单元。位图图像在放大时会失真，但却能够表达色彩丰富的图像效果。

矢量图使用直线和曲线来描述图形，图形元素是一些点、线、矩形、多边形、圆和弧线等，复杂图形是通过数学公式计算获得的。矢量图形的优点是在放大、缩小或旋转等情况下不会失真，缺点是难以表现色彩层次丰富的逼真图像效果。

思考与练习

7.9　请调研一下彩色向灰度转换的标准有哪些。

7.10　实例代码 12.1 如何遍历图像中每一个像素点？

7.11　Image.getpixel(x,y)返回值是什么？x、y 分别代表什么含义？

7.4　一二维数据的格式化和处理

要点： 数据组织存在维度，列表类型用于表示和处理一维和二维数据。

7.4.1　数据组织的维度

——数据还有维度？怎么没听说？百度都找不到。

——本书独家秘籍，一般人不告诉他。

计算机是能够根据指令操作数据的设备，因此操作数据是程序最重要的任务。除了单一数据类型（数字、浮点数等），更多的数据需要根据不同维度组织起来，以便进行管理和程序处理。根据数据的关系不同，数据组织可以分为一维数据、二维数据和高维数据。

一维数据由对等关系的有序或无序数据构成，采用线性方式组织，对应于数学

中的数组和集合等概念。例如，国际经济合作论坛 20 国集团（G20）的成员是对等关系，表示为一维数据，内容如下。无论采用任何方式分隔和表示，一维数据都具有线性特点。

```
中国、美国、日本、德国、法国、英国、意大利、加拿大、俄罗斯、欧盟、澳大利亚、南非、
阿根廷、巴西、印度、印度尼西亚、墨西哥、沙特阿拉伯、土耳其、韩国
```

二维数据，也称表格数据，由关联关系数据构成，采用表格方式组织，对应于数学中的矩阵，常见的表格都属于二维数据。例如，国家统计局发布的大中城市新建住宅价格指数是二维数据，摘录部分如表 7.12 所示。其中，表格说明部分（第一行）可以看作是二维数据的一个维度，也可以看作是数据外的说明。

表 7.12　2016 年 7 月部分大/中城市新建住宅价格指数

城　市	环　比	同　比	定　基
北京	101.5	120.7	121.4
上海	101.2	127.3	127.8
广州	101.3	119.4	120.0
深圳	102.0	140.9	145.5
沈阳	100.1	101.4	101.6

环比：上月=100；同比：上年同月=100；定基：2015 年=100。

高维数据由键值对类型的数据构成，采用对象方式组织，属于整合度更好的数据组织方式。高维数据在网络系统中十分常用，HTML、XML、JSON 等都是高维数据组织的语法结构。以描述本书作者的 JSON 格式为例，下面给出了这种高维数据的表示形式，其中，"本书作者"和后续内容通过冒号（:）形成一个键值对，每个内容中"姓氏"、"名字"和"单位"分别与后面的内容形成键值对。内容按照层级采用逗号和大括号组织起来。高维数据相比一维和二维数据能表达更加灵活和复杂的数据关系。

```
"本书作者" : [
                  { "姓氏" : "嵩",
                    "名字" : "天",
                    "单位" : "北京理工大学"    },
                  { "姓氏" : "礼",
                    "名字" : "欣",
                    "单位" : "北京理工大学"    },
                  { "姓氏" : "黄",
                    "名字" : "天羽",
                    "单位" : "北京理工大学"    }
              ]
```

数据包括文件存储和程序使用两个状态。存储不同维度的数据需要适合维度特点的文件存储格式，处理不同维度数据的程序需要使用相适应的数据类型或结构。因此，对于数据处理，需要考虑存储格式以及表示和读写等两个问题。

7.4.2 一二维数据的存储格式

一维数据是最简单的数据组织类型，有多种存储格式，常用特殊字符分隔，分隔方式如下。

（1）用一个或多个空格分隔，例如：

中国 美国 日本 德国 法国 英国 意大利

（2）用逗号分隔，注意，这里的逗号是英文输入法中的半角逗号，不是中文逗号，例如：

中国，美国，日本，德国，法国，英国，意大利

（3）用其他符号或符号组合分隔，建议采用不出现在数据中的特殊符号，例如：

中国; 美国; 日本; 德国; 法国; 英国; 意大利

二维数据由多条一维数据构成，可以看成是一维数据的组合形式。本书介绍一种国际通用的一二维数据存储格式：CSV 格式。这种格式十分简单，来源于使用逗号分隔的一维数据表示方式。

逗号分隔数值的存储格式叫做 CSV 格式（Comma-Separated Values，逗号分隔值），它是一种通用的、相对简单的文件格式，在商业和科学上广泛应用，尤其应用在程序之间转移表格数据。该格式的应用有如下一些基本规则。

（1）纯文本格式，通过单一编码表示字符。

（2）以行为单位，开头不留空行，行之间没有空行。

（3）每行表示一个一维数据，多行表示二维数据。

（4）以逗号（英文，半角）分隔每列数据，列数据为空也要保留逗号。

（5）对于表格数据，可以包含或不包含列名，包含时列名放置在文件第一行。

例如，表 7.12 中的二维数据采用 CSV 存储后的内容如下：

```
城市,环比,同比,定基
北京,101.5,120.7,121.4
上海,101.2,127.3,127.8
广州,101.3,119.4,120
深圳,102,140.9,145.5
沈阳,100.1,101.4,101.6
```

CSV 格式存储的文件一般采用.csv 为扩展名，可以通过 Windows 平台上的记事本或微软 Office Excel 工具打开，也可以在其他操作系统平台上用文本编辑工具打开。一般的表格数据处理工具（如微软 Office Excel 等）都可以将数据另存为或导出为 CSV 格式，用于不同工具间进行数据交换。

拓展： Python 的 csv 标准库

Python 提供了一个读写 csv 的标准库，可以通过 import csv 使用。csv 库包含操作 CSV 格式最基本的功能，csv.reader()和 csv.writer()。由于 CSV 格式十分简单，对于一般程序来说，建议程序员自己编写操作 CSV 格式的函数，这样更灵活和个性化。对于需要运行在复杂环境或商业使用的程序，建议采用 csv 标准库。

7.4.3　一二维数据的表示和读写

CSV 文件的每一行是一维数据，可以使用 Python 中的列表类型表示，整个 CSV 文件是一个二维数据，由表示每一行的列表类型作为元素，组成一个二维列表。例如，表 7.12 中的数据采用列表表示如下，对应代码见微实例 7.8。

```
[
    ['城市', '环比', '同比', '定基\n'],
    ['北京', '101.5', '120.7', '121.4\n'],
    ['上海', '101.2', '127.3', '127.8\n'],
    ['广州', '101.3', '119.4', '120.0\n'],
    ['深圳', '102.0', '140.9', '145.5\n'],
    ['沈阳', '100.1', '101.4', '101.6\n'],
]
```

源代码 7-9：导入 CSV 格式数据到列表



【微实例 7.8】导入 CSV 格式数据到列表。

将表 7.12 中的二维数据通过微软 Office Excel 等工具录入，另存成文件 price2016.csv，或直接使用本书附带文件，操作 CSV 文件的微实例代码如下：

微实例 7.8　　m7.8GetCSV2List.py

```
1  fo = open("price2016.csv", "r")
2  ls = []
3  for line in fo:
4      line = line.replace("\n","")
5      ls.append(line.split(","))
6  print(ls)
7  fo.close()
```

需要注意的是，以 split(",")方法从 CSV 文件中获得内容时，每行最后一个元素后面包含了一个换行符（"\n"）。对于数据的表达和使用来说，这个换行符是多余的，可以通过使用字符串的 replace()方法将其去掉，如第 4 行。

微实例 7.8 从 CSV 文件中一次性读入全部数据写入列表，之后，在程序内部使用列表即可表达数据，这种一次性读入方式适合一部分应用。另有一部分应用并不需要将数据全部读入程序再操作，可以逐行读取 CSV 文件，逐行运算处理，这种情况仅使用普通列表即可。

【微实例 7.9】逐行处理 CSV 格式数据。

从 CSV 文件中读取数据，去掉内容中的逗号，打印到屏幕。使用 price2016.csv 文件，代码如下：

源代码 7-10：逐行处理 CSV 格式数据



微实例 7.9　　m7.9GetCSVbyLine.py

```
1  fo = open("price2016.csv", "r")
2  ls = []
3  for line in fo:
4      line = line.replace("\n","")
5      ls = line.split(",")
6      lns = ""
7      for s in ls:
8           lns += "{}\t".format(s)
9      print(lns)
10 fo.close()
```

运行后的输出结果如下：

```
>>>
城市 环比 同比 定基
北京 101.5    120.7    121.4
上海 101.2    127.3    127.8
广州 101.3    119.4    120.0
深圳 102.0    140.9    145.5
沈阳 100.1    101.4    101.6
```

对于 Python 列表变量保存的一维数据结果，可以用字符串的 join()方法组成逗号分隔形式再通过文件的 write()方法存储到 CSV 文件中，具体过程参考微实例 7.10。其中，",".join(ls)生成一个新的字符串，它由字符串","分隔列表 ls 中的元素形成。

【微实例 7.10】一维数据写入 CSV 文件。

将一维数据['北京', '101.5', '120.7', '121.4']写入 price2016bj.csv 文件，代码如下：

源代码 7-11：一维数据写入 CSV 文件



微实例 7.10　　m7.10WriteD1toCSV.py

```
1  fo = open("price2016bj.csv", "w")
2  ls = ['北京', '101.5', '120.7', '121.4']
3  fo.write(",".join(ls)+ "\n")
4  fo.close()
```

对于列表中存储的二维数据，可以通过循环写入一维数据的方式写入 CSV 文件，参考代码样式如下：

```
for row in ls:
    <输出文件>.write(",".join(row)+"\n")
```

【微实例 7.11】二维数据写入 CSV 文件。

读入 price2016.csv 文件，将其中的数据读出，将数字部分计算百分比后输出到 price2016out.csv 文件。输出的 CSV 文件内容如下：

城市,环比,同比,定基
北京,1.0%,1.2%,1.2%
上海,1.0%,1.3%,1.3%
广州,1.0%,1.2%,1.2%
深圳,1.0%,1.4%,1.5%
沈阳,1.0%,1.0%,1.0%

整个程序分为 3 个部分：首先将原始文件中的数据全部导入，用列表方式表示；第二，对列表中的元素逐行判断，对浮点数值进行百分比运算，运算结果写回列表；第三，将更新后的列表输出新的 CSV 文件，代码如下：

微实例 7.11　　　　m7.11WriteD2toCSV.py

源代码 7-12：二维数据写入 CSV 文件



```
1   fr = open("price2016.csv", "r")
2   fw = open("price2016out.csv", "w")
3   ls = []
4   for line in fr:     #将 CSV 文件中的二维数据读入到列表变量
5       line = line.replace("\n","")
6       ls.append(line.split(","))
7   for i in range(len(ls)):  #遍历列表变量计算百分数
8       for j in range(len(ls[i])):
9           if ls[i][j].replace(".","").isnumeric():
10              ls[i][j] = "{:.2}%".format(float(ls[i][j])/100)
11  for row in ls:   #将列表变量中的两位数据输出到 CSV 文件
12      print(row)
13      fw.write(",".join(row)+"\n")
14  fr.close()
15  fw.close()
```

微程序 7.11 中第 9 行代码用于判断一个字符串是否类似"101.5"由数字或小数点构成。由于 Python 中没有单个函数能够直接判断，因此，通过 replace()方法将其中可能的小数点去掉，再通过 isnumeric()方法判断其余字符是否都是数字。这是一种不完备的判断方式，但在该例的应用背景下可以使用。

思考与练习

7.12　请描述数据维度的含义。
7.13　JSON 是一种什么样的数据格式？
7.14　思考 CSV 格式能否支持高维数据表示。

阶段测试 7-2：Python 数据维度理解小测验



7.5　实例 13：CSV 格式的 HTML 展示

要点： 这是一个将 CSV 格式转换为 HTML 展示的实例。

一个常见的需求是，用更直观的 HTML 方式通过浏览器展示 CSV 格式数据集。学习这个实例后，读者将具备处理一二维数据在存储、读取、操作、写入及展示方面的全套能力，操作基本数据不再是问题。

由于 CSV 主要存储二维表格数据，这个例子生成的 HTML 也是二维表格样式，与 CSV 数据直接对应。表 7.12 中的数据在运行后的效果如图 7.9 所示。

2016年7月部分大中城市新建住宅价格指数

城市	环比	同比	定基
北京	101.5	120.7	121.4
上海	101.2	127.3	127.8
广州	101.3	119.4	120.0
深圳	102.0	140.9	145.5
沈阳	100.1	101.4	101.6

图 7.9 二维数据的 HTML 展示

HTML（HyperText Markup Language）是超文本标记语言，它是专门为 Web（网页）显示创建的语言。HTML 语言本质上是键值对的标记，它采用<key>value</key>的方式表达键 key 对应的值 value。一个独立的 HTML 文件至少由<html>HTML 内容</html>构成，其中，内容部分由<body>body 内容</body>构成。HTML 语言具有完善庞大的语法体系，这里仅介绍与表格展示相关的内容。首先看一下 HTML 代码的格式，图 7.9 展示二维数据对应的 HTML 完整代码如下，该文件名称为 CSV2HTML.html。

文件名： CSV2HTML.html

```
1   <!DOCTYPE HTML>
2   <html>
3   <body>
4   <meta charset=utf-8>
5   <h2 align=center>2016 年 7 月部分大中城市新建住宅价格指数</h2>
6   <table border='1' align=center width=70%>
7   <tr bgcolor='orange'>
8   <th width="25%">城市</th>
9   <th width="25%">环比</th>
10  <th width="25%">同比</th>
11  <th width="25%">定基</th>
12  </tr>
13  <tr><td>北京</td><td>101.5</td><td>120.7</td><td>121.4</td></tr>
14  <tr><td>上海</td><td>101.2</td><td>127.3</td><td>127.8</td></tr>
15  <tr><td>广州</td><td>101.3</td><td>119.4</td><td>120.0</td></tr>
16  <tr><td>深圳</td><td>102.0</td><td>140.9</td><td>145.5</td></tr>
17  <tr><td>沈阳</td><td>100.1</td><td>101.4</td><td>101.6</td></tr>
18  </table>
19  </body>
20  </html>
```

HTML 语法中，由<table>表格内容</table>形成的是一个表格，<tr>表格的一行</tr>表示表格的一行。<tr>标签中<th>表头列</th>表示表格表头的一列；<td>内容列</td>表示表格内容的一列。

拓展： Web 前端开发

Web 前端开发指开发基于 HTML 的展示效果，经历过静态网页制作、动态网页制作和 Web 2.0 开发等几个阶段。Web 前端开发主要采用 HTML 5、CSS、JavaScript 等语言，这些语言由 W3C（万维网联盟）进行规范化并制定标准。目前大量手机应用的界面也采用 Web 开发，相比用安卓系统中的 Java 语言和 iOS 系统中的 Object C 语言，Web 开发的手机应用能够在不同手机操作系统上运行。

源代码 7-13：CSV 格式的 HTML 展示



将 CSV 文件转换成 HTML 文件分为 3 个步骤：首先，读入 CSV 文件，获得文件数据；其次，对数据进行格式处理和转换；最后，输出与上述代码相同的 HTML 格式文件。CSV2HTML.html 文件中除了数据之外都是格式化的规则字符串。这个程序的设计思路是不管 HTML 样式如何，只要替换其中的数据，就能够显示不同数据对应的表格。实例代码 13.1 如下。

实例代码 13.1　　　e13.1csv2html.py

```
1   #e13.1csv2html.py
2   seg1 = '''
3   <!DOCTYPE HTML>\n<html>\n<body>\n<meta charset=gb2312>
4   <h2 align=center>2016 年 7 月部分大中城市新建住宅价格指数</h2>
5   <table border='1' align="center" width=70%>
6   <tr bgcolor='orange'>\n '''
7   seg2 = "</tr>\n"
8   seg3 = "</table>\n</body>\n</html>"
9   def fill_data(locls):
10      seg = '<tr><td align="center">{}</td><td align="center">\
11      {}</td><td align="center">{}</td><tdalign="center">\
12      {}</td></tr>\n'.format(*locls)
13      return seg
14  fr = open("price2016.csv", "r")
15  ls = []
16  for line in fr:
17      line = line.replace("\n","")
18      ls.append(line.split(","))
19  fr.close()
20  fw = open("price2016.html", "w")
21  fw.write(seg1)
22  fw.write('<th width="25%">{}</th>\n<th
23  width="25%">{}</th>\n<th width="25%">{}</th>\n<th
24   width="25%">{}</th>\n'.format(*ls[0]))
25  fw.write(seg2)
```

```
26  for i in range(len(ls)-1):
27      fw.write(fill_data(ls[i+1]))
28  fw.write(seg3)
29  fw.close()
```

实例代码 13.1 将 CSV 文件中的数据读入列表 ls，然后通过格式化字符串方法将 ls 中的内容写入 HMTL 文件。运行结果如图 7.9 所示。

思考与练习

7.15 请调研 HTML 语言的基本语法形式。

7.16 实例代码 13.1 中的第 17 行的作用是什么？

7.17 如何修改实例代码 13.1，使得生成的 HTML 表格的表头是淡蓝色？

7.6 高维数据的格式化

要点： 键值对是高维数据的特征，采用 JSON 格式对高维数据进行表达和存储。

与一维、二维数据不同，高维数据能展示数据间更为复杂的组织关系。为了保持灵活性，表示高维数据不采用任何结构形式，仅采用最基本的二元关系，即键值对。

万维网（WWW）是一个复杂的数据组织体系，它通过 HTML 方式链接并展示不同类型数据内容，采用 XML 或 JSON 格式表达键值对，形成数据间复杂的结构关系。万维网是高维数据最成功的典型应用。

JSON 格式可以对高维数据进行表达和存储。JSON（JavaScript Object Notation）是一种轻量级的数据交换格式，易于阅读和理解。JSON 格式表达键值对<key, value>的基本格式如下，键值对都保存在双引号中：

```
"key" : "value"
```

当多个键值对放在一起时，JSON 有如下一些约定。

（1）数据保存在键值对中。

（2）键值对之间由逗号分隔。

（3）大括号用于保存键值对数据组成的对象。

（4）方括号用于保存键值对数据组成的数组。

以“本书作者”JSON 数据为例。

```
"本书作者" : [
                {   "姓氏" : "嵩",
                    "名字" : "天",
```

```
                "单位" : "北京理工大学"   },
            {   "姓氏" : "礼",
                "名字" : "欣",
                "单位" : "北京理工大学"   },
            {   "姓氏" : "黄",
                "名字" : "天羽",
                "单位" : "北京理工大学"   }
        ]
```

首先它是一个键值对，由“本书作者”与内容组成；由于存在 3 个作者，作者之间采用逗号分隔，作者之间是对等关系，形成一个数组，采用方括号分隔；每个作者是一个对象，采用大括号组织，因为对象中包括作者的姓氏、名字和单位，每一项都是一个键值对，对应作者的一个属性。

采用对象、数组方式组织起来的键值对可以表示任何结构的数据，这为计算机组织复杂数据提供了极大的便利。目前，万维网上使用的高维数据格式主要是 JSON 和 XML，本书建议采用 JSON 格式。格式化高维数据采用 Python 语言的标准库 json。

拓展： XML

XML（Extensible Markup Language，可扩展标记语言）是 W3C 推荐的一种开放标准，它用来为 Internet 上传送及携带数据信息提供标准格式。简单地说，XML 格式需要成对的标签表示键值对。“本书作者”的 XML 描述如下。

```
<本书作者>
    <姓氏>嵩</姓氏><名字>天</名字><单位>北京理工大学</单位>
    <姓氏>礼</姓氏><名字>欣</名字><单位>北京理工大学</单位>
    <姓氏>黄</姓氏><名字>天羽</名字><单位>北京理工大学</单位>
</本书作者>
```

XML 和 JSON 都可以表达高维数据，但 XML 对 key 值要存储两次（<key></key>），而 JSON 只需要存储一次，且在数据交换时产生更少的网络带宽和存储需求，因此相比 XML 更为常用。

思考与练习

7.18　思考键值对对高维数据构建的意义。

7.19　JSON 格式中大括号和中括号的作用是什么？

7.20　思考 JSON 如何支持一二维数据表示。

7.7　模块 6：json 库的使用

要点： json 库是处理 JSON 格式的 Python 标准库。

7.7.1 json 库概述

json 库是处理 JSON 格式的 Python 标准库，导入方式如下：

```
>>> import json
```

json 库主要包括两类函数：操作类函数和解析类函数。操作类函数主要完成外部 JSON 格式和程序内部数据类型之间的转换功能；解析类函数主要用于解析键值对内容。json 格式包括对象和数组，用大括号{}和方括号[]表示，分别对应键值对的组合关系和对等关系。使用 json 库时需要注意 json 格式的“对象”和“数组”概念与 Python 语言中“字典”和“列表”的区别和联系。一般来说，JSON 格式的对象将被 json 库解析为字典，JSON 格式的数组将被解析为列表。

7.7.2 json 库解析

json 库包含两个过程：编码（encoding）和解码（decoding）。编码是将 Python 数据类型变换成 JSON 格式的过程，解码是从 JSON 格式中解析数据对应到 Python 数据类型的过程。本质上，编码和解码是数据类型序列化和反序列化的过程。

拓展： 序列化

> 序列化是指将对象数据类型转换为可以存储或网络传输格式的过程，传输格式一般为 JSON 或 XML。反序列化指从存储区域中将 JSON 或 XML 格式读出并重建对象的过程。JSON 序列化与反序列化的过程分别是编码和解码。

表 7.13 列出了 json 库的 4 个操作类函数，其中 dumps()和 loads()分别对应编码和解码功能。

表 7.13 json 库的操作类函数（共 4 个）

函数	描述
json.dumps(obj, sort_keys=False, indent=None)	将 Python 的数据类型转换为 JSON 格式，编码过程
json.loads(string)	将 JSON 格式字符串转换为 Python 的数据类型，解码过程
json.dump(obj, fp, sort_keys=False, indent=None)	与 dumps()功能一致，输出到文件 fp
json.load(fp)	与 loads()功能一致，从文件 fp 读入

json.dumps()中的 obj 可以是 Python 的列表或字典类型，当输入字典类型时，dumps()函数将其变为 JSON 格式字符串。默认生成的字符串是顺序存放的，sort_keys 可以对字典元素按照 key 进行排序，控制输出结果。indent 参数用于增加数据缩进，使得生成的 JSON 格式字符串更具有可读性。

```
>>>dt = {'b':2,'c':4,'a':6}
```

```
>>>s1 = json.dumps(dt)  #dumps 返回 JSON 格式的字符串类型
>>>s2 = json.dumps(dt,sort_keys=True,indent=4)
>>>print(s1)
{"c": 4, "a": 6, "b": 2}
>>>print(s2)
{
   "a": 6,
   "b": 2,
   "c": 4
}
>>>print(s1==s2)
False
>>>dt2 = json.loads(s2)
>>>print(dt2, type(dt2))
{'c': 4, 'a': 6, 'b': 2} <class 'dict'>
```

尽管 json 库还有很多丰富的功能，但是 json 库一般用于 JSON 格式和其他类型格式转换，所以，掌握基本用法即可。

程序练习 7-3：十分钟学 json 库



思考与练习

7.21　json 库的 dumps()函数的 sort_keys 参数有何作用？

7.22　判断题：json 库的 dumps()函数将 Python 字典类型变成字符串。

7.23　判断题：json 库的 dumps()函数将 Python 列表类型变成字符串。

7.8　实例 14：CSV 和 JSON 格式相互转换

要点： 这是一个 CSV 和 JSON 格式相互转换的实例。

CSV 格式常用于一二维数据表示和存储，它是一种纯文本形式存储表格数据的表示方式。JSON 也可以表示一二维数据。在网络信息传输中，可能需要统一表示方式，因此，需要在 CSV 和 JSON 格式间进行相互转换。

以 price2016.csv 作为输入，希望输出的 JSON 格式对应于每行数据如下：

```
    "同比": "120.7",
    "城市": "北京",
    "定基": "121.4",
    "环比": "101.5"
```

由于 CSV 格式和 JSON 格式处理在本章中都有介绍，这里仅给出转换的完整程序。

将 CSV 格式转换成 JSON 格式的代码如下：

源代码 7-14：CSV 格式向 JSON 格式的转换



实例代码 14.1　　　　e14.1csv2json.py

```
1   #e14.1csv2json.py
2   import json
3   fr = open("price2016.csv", "r")
4   ls = []
5   for line in fr:
6       line = line.replace("\n","")
7       ls.append(line.split(','))
8   fr.close()
9   fw = open("price2016.json", "w")
10  for i in range(1,len(ls)):
11      ls[i] = dict(zip(ls[0], ls[i]))
12  json.dump(ls[1:],fw, sort_keys=True, indent=4)
13  fw.close()
```

其中，zip()是一个内置函数，能够将两个长度相同的列表组合成一个关系对，例子如下，该函数非常适合于生成键值对。

```
>>>x = [1, 2, 3]
>>>y = ["a", "b", "c"]
>>>list(zip(x,y))
[(1, 'a'), (2, 'b'), (3, 'c')]
```

实例代码 14.1 生成文件中没有输出中文字符，原内容中的中文字符被替换成 Unicode 编码，部分内容如下：

```
[
    {
        "\u540c\u6bd4": "120.7",
        "\u57ce\u5e02": "\u5317\u4eac",
        "\u5b9a\u57fa": "121.4",
        "\u73af\u6bd4": "101.5"
    },
...
```

json 库默认采用 Unicode 编码处理非西文字符，主要为了避免网络传输中因编码方式不同带来的问题。可以通过在 dumps()函数中修改 ensure_ascii 参数默认值使 json 库输出中文字符。修改实例代码 14.1 中的第 12 行代码：

```
json.dump(ls[1:],fw, sort_keys=True, indent=4, ensure_ascii=False)
```

程序运行后的完整输出结果如下：

```
[
    {
        "同比": "120.7",
        "城市": "北京",
        "定基": "121.4",
        "环比": "101.5"
    },
    {
        "同比": "127.3",
        "城市": "上海",
        "定基": "127.8",
        "环比": "101.2"
    },
    {
        "同比": "119.4",
        "城市": "广州",
        "定基": "120",
        "环比": "101.3"
    },
    {
        "同比": "140.9",
        "城市": "深圳",
        "定基": "145.5",
        "环比": "102"
    },
    {
        "同比": "101.4",
        "城市": "沈阳",
        "定基": "101.6",
        "环比": "100.1"
    },
    {
        "同比": "",
        "城市": "",
        "定基": "",
        "环比": ""
    }
]
```

将二维 JSON 格式数据转换成 CSV 格式数据的代码如下，供读者参考。

源代码 7-15:
JSON 格式向 CSV 格式的转换



实例代码 14.2　　　　e14.2json2csv.py

```
1   #14.2json2csv.py
2   import json
3   fr = open("price2016.json", "r")
4   ls = json.load(fr)
5   data = [ list(ls[0].keys()) ]
6   for item in ls:
7       data.append(list(item.values()))
8   fr.close()
9   fw = open("price2016_from_json.csv", "w")
10  for item in data:
11      fw.write(",".join(item) + "\n")
12  fw.close()
```

拓展: Python 的数据类型转换

表 7.14 给出了 Python 数据类型转换函数，请读者比较参考。

表 7.14　Python 的数据类型转换函数（共 13 个）

函　　数	描　　述
int(x [,base])	将字符串 x 转换为一个整数
float(x)	将字符串 x 转换为一个浮点数
complex(real [,imag])	根据 real 和 imag 创建一个浮点数
str(x)	将对象 x 转换为字符串
repr(obj)	将对象 obj 当作 Python 语句执行，返回结果的字符串形式
eval(str)	计算字符串中的有效 Python 表达式，返回结果
tuple(s)	将序列 s 转换为一个元组
list(s)	将序列 s 转换为一个列表
chr(x)	将一个整数转换为一个字符
unichr(x)	将一个整数转换为 Unicode 字符
ord(x)	将一个字符转换为它的整数值
hex(x)	将一个整数转换为一个十六进制字符串
oct(x)	将一个整数转换为一个八进制字符串

思考与练习

7.24　如何通过 json 库的 dumps()函数输出中文字符？

7.25　zip()函数的作用是什么？

7.26　思考列表和字典在各维度数据处理中的应用。

本 章 小 结

本章主要介绍了文件的输入输出操作框架，介绍了 PIL 库并使用 PIL 库演示了

字符画绘制实例，进一步介绍了数据的维度概念和多维数据的格式化处理方法，演示了 JSON 和 CSV 格式相互转化以及 CSV 的 HTML 格式展示等方法。

程序练习题



7.1　Python 源文件改写。编写一个程序，读取一个 Python 源程序文件，将文件中所有除保留字外的小写字母换成大写字母，生成后的文件要能够被 Python 解释器正确执行。

7.2　图像文件压缩。使用 PIL 库对图片进行等比例压缩，无论压缩前文件大小如何，压缩后文件小于 10 KB。

7.3　中文字符画。参考实例 12，编写程序合理选取中文字符构造字符表，生成中文字符画。

7.4　CSV 解析。改编实例 14，使得对 CSV 的转换能够识别并保留数据内部的逗号。

7.5　制作英文学习词典。编写程序制作英文学习词典，词典有 3 个基本功能：添加、查询和退出。程序读取源文件路径下的 txt 格式词典文件，若没有就创建一个。词典文件存储方式为“英文单词 中文单词”，每行仅有一对中英释义。程序会根据用户的选择进入相应的功能模块，并显示相应的操作提示。当添加的单词已存在时，显示“该单词已添加到字典库”；当查询的单词不存在时，显示“字典库中未找到这个单词”。用户输入其他选项时，提示“输入有误”。

7.6　修改程序练习题 7.5 的程序，使其能够对单词添加多重释义，不同释义用逗号分开。

第三部分　运用 Python 语言

MOOC 课程：
中国大学 MOOC
“Python 语言程序设计”课程



本部分从程序设计方法角度讲解利用 Python 语言解决完整实际问题的过程和方法，分别在图形艺术、科学计算、数据处理和网络应用等方面给出具体实例并介绍各领域专业第三方库的使用，使读者理解 Python 语言的“模块编程”思想。本部分的学习目标是，编写 100 行左右“炫酷（库）”的 Python 程序。

本部分包括 5 章内容（第 8~10 章、附录 B、附录 C），分别如下：

第 8 章主要讲解程序设计方法学，包括计算思维、自顶向下、自底向上、Python 第三方库安装和使用，这里推荐读者了解计算生态和模块编程思想。

第 9 章主要面向科学计算和可视化，讲解多维数据运算第三方库 numpy 和科学计算可视化库 matplotlib，重点讲解绘制坐标系和雷达图的方法。

第 10 章主要面向互联网，讲解网络爬虫设计原理和网页解析方法，介绍该领域最优秀的 requests 库和 beautifulsoup4 库，在爬取内容同时讲解提交内容方法。

附录 B 主要面向图像和艺术设计，讲解图形用户界面的编写方法及图形艺术的设计与实现，重点介绍最优秀的第三方库 PyQt5 和标准库 turtle。

附录 C 主要面向数据处理和挖掘，讲解利用优秀的数据挖掘第三方库进行聚类、分类和回归等程序设计，重点讲解 sklearn 库及 3 个重要的数据挖掘算法的使用。

鉴于读者兴趣不同，第 9~10 章和附录 B、C 的内容可以采用 4 选 2 或者 4 选 3 方式学习，这 4 章内容中只有附录 C 内容以第 9 章部分知识为基础，其他内容无交叉。

第 8 章　程序设计方法论

电子教案 8-1：
程序设计方法论



人生苦短，请用 Python。

Life is short. You need Python.

——布鲁斯・埃克尔（Bruce Eckel）

ANSI/ISO C++标准委员会发起者之一

学习目标

（1）了解计算思维的概念。
（2）掌握自顶向下的设计方法。
（3）掌握自底向上的执行过程。
（4）了解计算生态和模块编程思想。
（5）掌握 Python 第三方库的安装方法。
（6）掌握 Python 源文件的打包方法。

难得学会了 Python 编程，但每次执行代码都要通过 IDLE 或命令行。满怀喜悦地想跟周围朋友分享代码，却发现跨平台兼容出了问题，一万点伤害！

——有没有简单方法，既可以将程序打包为可执行文件又可以在多平台下良好兼容？

——当然有，本章将介绍神器 pyinstaller 库，教你如何制作程序小包裹。

人生苦短，快用 Python，盲目苦干不如合理偷懒。

8.1 计算思维

要点： 计算思维是人类科学思维活动的重要组成部分，与逻辑思维和实证思维同等重要。

2006 年，时任美国卡内基-梅隆大学计算机系主任的周以真（Jeannette M. Wing）教授提出了计算思维（Computational Thinking）概念，这个概念第一次从思维层面阐述了运用计算机科学的基础概念求解问题、设计系统和理解人类行为的过程。程序设计是实践计算思维的重要手段，本书之前的各个实例虽然问题不同，但都采用了同一种解决思路：抽象实际问题的计算特性，利用计算机求解。计算思维的本质是抽象（Abstraction）和自动化（Automation）。

计算思维是人类科学思维活动的重要组成部分。人类在认识世界、改造世界过程中表现出 3 种基本的思维特征：以实验和验证为特征的实证思维，以物理学科为代表；以推理和演绎为特征的逻辑思维，以数学学科为代表；以设计和构造为特征的计算思维，以计算机学科为代表。理解计算思维，除了认识概念本身，还要清晰地认识它的时代特性。

计算思维并非天生就存在且清晰，人类探索自然几千年中仅有十分“朦胧”的计算概念。即使 1946 年第一台计算机 ENIAC 的诞生，也并没有让人类对计算有重新认识，更难从思维角度理解和认识计算。然而，随后计算机技术的快速发展改变了人类的认知。摩尔定律和网络技术在极短时间内让计算机以极低的成本走入了人类日常生活，大量传统行业通过信息化改造大幅提升了效率，显著的变化让人类意识到计算机的强大力量。人类已经开始依赖计算机带来的丰富计算能力，思维方式也在逐渐演变。2006 年，周以真教授深刻阐述了计算思维的概念。可以看出，计算思维是计算机科学发展到一定程度而提出的，它是人类逐渐意识到计算机解决问题的强大能力后而自然产生的思维模式，具有显著的时代特性。从内涵角度讲，以“抽象和自动化”为特点的计算思维必然以当今计算机科学与技术的发展为前提，对计算思维的认识要与计算机科学发展阶段相适应。

在程序设计范畴，计算思维主要反映在理解问题的计算特性、将计算特性抽象为计算问题、通过程序设计语言实现问题的自动求解等几个方面。

拓展： ENIAC

ENIAC（Electronic Numerical Integrator And Computer，电子数字积分计算机）是世界上第一台通用计算机，于 1946 年诞生于美国宾夕法尼亚大学摩尔实验室。尽管 ENIAC 占地面积约 170 平方米，重达 30 t，它比当时最快的计算设备速度快 1 000 倍。这样惊人的计算性能在随后半个多世纪里改变了整个世界，原始创新极其重要。

思考与练习

8.1 计算思维的本质是什么？

8.2 简述通过计算思维解决问题的基本过程。

8.3 下列实例中是计算思维的应用的是（ ）。

A．通过多次的实验与统计，总结事件发生的规律

B．通过复杂的推导，验证了数学公式的正确性

C．高考中，算出了一道很难的数学题

D．对一类问题进行数学建模，并通过程序解决问题

8.2 实例15：体育竞技分析

要点： 这是一个模拟体育竞技并进行竞技分析的实例。

模拟是用来解决现实世界问题的重要手段和技术。计算机可以通过模拟现实世界的运行过程提供一般情况下无法获得的信息，使用计算机模拟解决问题的实例包括天气预测、飞机设计、电影特效、核试验甚至军事对抗等。如果不采用计算机模拟，这些应用则需要极其复杂的实施过程，往往代价巨大。即使很简单的模拟也可以揭示一些困难问题的本质规律。

体育竞技像其他竞技一样为合作、对抗和策略提供了一个施展的平台。体育竞技历史悠久，中国古代就有田忌赛马的故事。到了现代，高水平竞技活动中双方实力差距越来越小，胜负往往在毫厘间。因此，体育竞技分析作为挖掘并解释竞技背后规律和现象的手段逐渐成为了人们关注的重要问题。现实中的例子，一个朋友A，他非常喜欢打网球，多年经历和体育精神使A养成了一个习惯，他总喜欢和那些比他强一点的人比赛。为此，他经常被完胜，输掉了绝大多数比赛。A总是质疑这里面是运气原因、状态原因或不可控因素。真是这样吗？从直观感受来看，水平稍好的球员似乎会赢得稍多，比赛结果不会相差悬殊。A与其他人比赛仅有一些弱势，是不应该总被完胜的。这个生活中的直观例子可以反映深刻问题。排除一些主观因素，例如与A比赛的其他人也许水平比A高很多，只是他们不承认而已。本节将通过编写一个计算机程序来模拟体育竞技的某些属性环节，进而模拟上千场不同水平级别对手之间的比赛，揭示并分析体育竞技规律。

拓展： 模拟和仿真

模拟（simulation）是抽象原系统某些行为特征并用另一系统来表示这些特征的过程，通常用于设计初期的模型验证。仿真（emulation）则更进一步，需要模仿系统真实能做的事情，接收同样的数据，获得同样的结果，只不过实现的过程不同。仿真一般用于处理兼容性问题或在资源有限的条件下实现系统原型。

本节使用一种从各种球类比赛中抽象的一般规则，规则定义如下：两个球员在一个有 4 面边界的场地上用球拍击球。开始比赛时，其中一个球员首先发球。接下来球员交替击球，直到可以判定得分为止，这个过程称为回合。当一名球员未能进行一次合法击打时，回合结束。未能打中球的球员输掉这个回合。如果输掉这个回合的是发球方，那么发球权交给另一方；如果输掉的是接球方，则仍然由这个回合的发球方继续发球。总之，每回合结束，由赢得该回合的一方发球。球员只能在他们自己的发球局中得分。首先达到 15 分的球员赢得一局比赛。

在计算机模拟中，运动员的能力级别将通过发球方赢得本回合的概率来表示。因此，一个 0.6 概率的球员可以在他的发球局有 60%的可能性赢得 1 分。程序首先接收两个球员的水平值，然后利用这个值采用概率方法模拟多场比赛。程序最后会输出比赛运行结果。

该问题的 IPO 描述如下。

输入：两个球员（球员 A 和 B）的能力概率，模拟比赛的场次

处理：模拟比赛过程

输出：球员 A 和 B 分别赢得球赛的概率

抽象这个问题时，将球员失误、犯规等可能性一并考虑在能力概率中，在每局比赛中，球员 A 先发球。一个期望的输出结果如下。

模拟比赛数量：500

球员 A 获胜场次：268（53.6%）

球员 B 获胜场次：232（46.4%）

体育竞技分析程序需要面对不确定事件。一个球员赢得了 50%的发球权，并不意味着剩下的每一局他都是胜利者，这更像是掷硬币。

解决体育竞技分析问题似乎与之前所解决的问题有所不同，因为其处理过程并不是仅靠一个算法完成，而是需要稍微复杂的程序结构。虽然该问题在 Python 中实现并不复杂，但对该问题设计的讨论有助于理解程序中的一些重要方法。8.3 节将结合这个例子介绍自顶向下的设计方法和自底向上的执行过程。

思考与练习

8.4　思考体育竞技分析实例中体现的计算思维思想。

8.5　思考还有哪些应用使用了计算机模拟。

8.3　自顶向下和自底向上

> **要点：** 程序需要采用自顶向下的设计方法，采用自底向上的执行方法。

一个解决复杂问题行之有效的方法被称作自顶向下的设计方法，其基本思想是以一个总问题开始，试图把它表达为很多小问题组成的解决方案。再用同样的技术依次攻破每个小问题，最终问题变得非常小，以至于可以很容易解决。然后只需把

所有的碎片组合起来，就可以得到一个程序。

8.3.1　自顶向下设计

1. 顶层设计

自顶向下设计中最重要的是顶层设计。以体育竞技分析为例，可以从问题的 IPO 描述开始。大多数程序都可以将 IPO 描述直接用到程序结构设计中，体育竞技分析从用户处得到模拟参数，最后输出结果。下面是一个基础设计的 4 个步骤。

步骤 1：打印程序的介绍性信息。

步骤 2：获得程序运行需要的参数，即 probA、probB、n。

步骤 3：利用球员 A 和 B 的能力值 probA 和 probB，模拟 n 次比赛。

步骤 4：输出球员 A 和 B 获胜比赛的场次及概率。

这个基础设计从 IPO 描述获得，可以作为自顶向下设计的顶层设计。

步骤 1 输出一些介绍信息，针对提升用户体验十分有益。下面是这个步骤的 Python 代码，顶层设计一般不写出具体代码，仅给出函数定义，其中，printIntro()函数打印一些必要的说明。

```
1  def main():
2      printIntro()
```

步骤 2 获得用户输入。通过函数将输入语句及输入格式等细节封装或隐藏，只需要假设程序如果调用了 getInputs()函数即可获取变量 probA、probB 和 n 的值。这个函数必须为主程序返回这些值，截至第 2 步，全部代码如下：

```
1  def main():
2      printIntro()
3      probA, probB, n = getInputs()
```

步骤 3 需要使用 probA、probB 模拟 n 场比赛。此时，可以采用步骤 2 的类似方法，设计一个 simNGames()函数来模拟 n 场比赛，并返回结果。按照体育竞技问题的要求，该函数需要模拟比赛，并给出球员 A 和球员 B 赢得比赛的结果。截止步骤 3，程序的 Python 代码如下：

```
1  def main():
2      printIntro()
3      probA, probB, n = getInputs()
4      winsA, winsB = simNGames(n, probA, probB)
```

步骤 4 输出结果，设计思想类似，仍然只规划功能和函数，代码如下：

```
1 def main():
2     printIntro()
3     probA, probB, n = getInputs()
4     winsA, winsB = simNGames(n, probA, probB)
5     printSummary(winsA, winsB)
```

至此，体育竞技分析问题的程序框架已经清晰，但这仅是框架，main()函数并没有做什么。原问题被划分为了 4 个独立的函数：printIntro()、getInputs()、simNGames()和 printSummary()。这些函数的名称、输入参数和预期返回值都已经确定。这个分解过程十分有益，因为它让程序员在这一步不必关心具体细节而专心考虑程序的结构设计。

2. 第 *n* 层设计

经过顶层设计，main()函数成为体育竞技分析的顶层结构，上述设计可以表示为图 8.1，其中每层按照从左至右的顺序执行，每个函数用一个矩形表示，连接两个矩形的线表示上面函数对下面函数的调用关系。在信息流方面，箭头和注释表示函数之间的输入和输出。

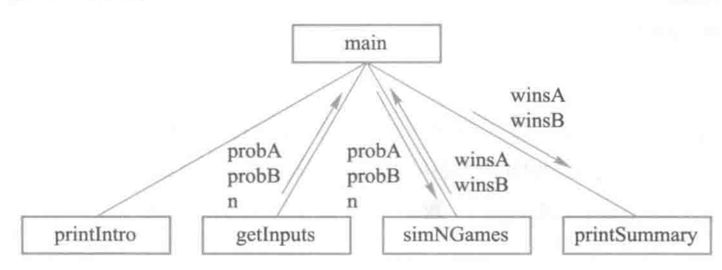


图 8.1 体育竞技分析程序结构图：顶层设计

每层设计中，参数和返回值如何设计是重点，其他细节可以暂时忽略。确定事件的重要特征而忽略其他细节过程称为抽象。抽象是一种基本设计方法，自顶向下的设计过程可以看作是发现功能并抽象功能的过程。自顶向下设计的第二阶段是实现或进一步抽象第 2 层函数。

printIntro()函数应该输出一个程序介绍，这个功能的 Python 代码如下，这个函数由 Python 基本表达式组合，不增加或改变程序结构。

```
1 def printIntro():
2     print("这个程序模拟两个选手 A 和 B 的某种竞技比赛")
3     print("程序运行需要 A 和 B 的能力值（以 0 到 1 之间的小数表示）")
```

getInputs()函数根据提示得到 3 个需要返回主程序的值，代码如下：

```
1  def getInputs():
2      a = eval(input("请输入选手 A 的能力值(0-1): "))
3      b = eval(input("请输入选手 B 的能力值(0-1): "))
4      n = eval(input("模拟比赛的场次: "))
5      return a, b, n
```

simNGames()函数是整个程序的核心，其基本思路是模拟 *n* 场比赛，并跟踪记录每个球员赢得了多少比赛。“模拟 *n* 场比赛”直观感受像一个计数循环，而跟踪记录获胜场次更像计数过程。这是一个相当直观且粗粒度的设计，类似顶层设计，其 Python 代码如下：

```
1  def simNGames(n, probA, probB):
2      winsA, winsB = 0, 0
3      for i in range(n):
4          scoreA, scoreB = simOneGame(probA, probB)
5          if scoreA > scoreB:
6              winsA += 1
7          else:
8              winsB += 1
9      return winsA, winsB
10
```

代码中设计了 simOneGame()函数，用于模拟一场比赛，这个函数需要知道每个球员的概率，返回两个球员的最终得分，图 8.2 给出了这个设计对整体结构的更新。

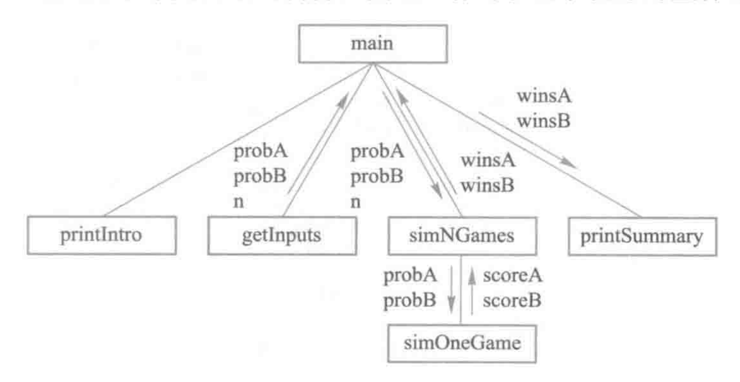


图 8.2　体育竞技分析程序结构图：第二阶段

接下来需要实现 simOneGame()函数。为了模拟一场比赛，需要根据比赛规则来编写代码，两个球员 A 和 B 持续对攻直至比赛结束。可以采用无限循环结构直到比赛结束条件成立。同时，需要跟踪记录比赛得分，保留发球局标记，总之，尽可能详细地模拟比赛过程。在模拟比赛的循环中，需要考虑单一的发球权和比分问题，

通过随机数和概率，可以确定发球方是否赢得了比分（random() < prob）。如果球员 A 发球，那么需要使用 A 的概率，接着根据发球结果，更新是球员 A 得分还是将球权交给球员 B。该函数的代码如下：

```
1  def simOneGame(probA, probB):
2      scoreA, scoreB = 0, 0
3      serving = "A"
4      while not gameOver(scoreA, scoreB):
5          if serving == "A":
6              if random() < probA:
7                  scoreA += 1
8              else:
9                  serving="B"
10         else:
11             if random() < probB:
12                 scoreB += 1
13             else:
14                 serving="A"
15     return scoreA, scoreB
```

这里进一步设计了 gameOver()函数，用来表示一场比赛结束的条件，对于不同体育比赛结束条件可能不同，封装该函数有助于简化根据不同规则修改函数的代价，提高代码可维护性。gameOver()函数跟踪分数变化并在比赛结束时返回 True，未结束则返回 False。然后继续循环的其余部分。图 8.3 是程序新的结构图。

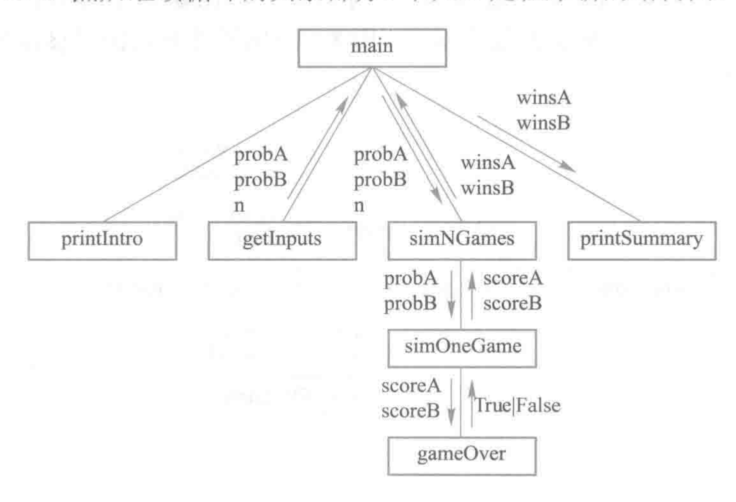


图 8.3 体育竞技分析程序结构图：第三阶段

根据比赛规则，当任意一个球员分数达到 15 分时比赛结束。gameOver()函数实现代码如下：

```
1  def gameOver(a,b):
2      return a==15 or b==15
```

最后是 printSummary()函数，其 Python 代码如下：

```
1  def printSummary(winsA, winsB):
2      n = winsA + winsB
3      print("竞技分析开始，共模拟{}场比赛".format(n))
4      print("选手 A 获胜{}场比赛，占比{:0.1%}".format(winsA, winsA/n))
5      print("选手 B 获胜{}场比赛，占比{:0.1%}".format(winsB, winsB/n))
```

将上述所有代码放在一起，形成了实例代码 15.1。

源代码 8-1：体育竞技分析



实例代码 15.1　　e15.1MatchAnalysis.py

```
1  #e15.1MatchAnalysis.py
2  #e15.1MatchAnalysis.py
3  from random import random
4  def printIntro():
5      print（"这个程序模拟两个选手 A 和 B 的某种竞技比赛"）
6      print（"程序运行需要 A 和 B 的能力值（以 0 到 1 之间的小数表示）"）
7  def getInputs():
8      a=eval（input（"请输入选手 A 的能力值（0-1）："））
9      b=eval（input（"请输入选手 B 的能力值（0-1）："））
10     n=eval（input（"模拟比赛的场次："））
11     return a, b, n
12 def simNGames（n, probA, probB）:
13     winsA, winsB=0,0
14     for i in range(n):
15         scoreA, scoreB=simOneGame（probA, probB）
16         if scoreA>scoreB:
17             winsA += 1
18         else:
19             winsB += 1
20     return winsA, winsB
21 def gameOver(a, b):
22     return a==15 or b==15
23 def simOneGame(probA, probB):
24     scoreA, scoreB = 0, 0
25     serving = "A"
26     while not gameOver(scoreA, scoreB):
27         if serving == "A":
```

```
28                if random() < probA:
29                    scoreA += 1
30                else:
31                    serving="B"
32            else:
33                if random() < probB:
34                    scoreB += 1
35                else:
36                    serving="A"
37        return scoreA, scoreB
38    def printSummary(winsA, winsB):
39        n=winsA+winsB
40        print("竞技分析开始，共模拟{}场比赛".format(n))
41        print("选手A获胜{}场比赛，占比{:0.1%}".format(winsA, winsA/n))
42        print("选手B获胜{}场比赛，占比{:0.1%}".format(winsB, winsB/n))
43    def main():
44        printIntro()
45        probA, probB, n = getInputs()
46        winsA, winsB = simNGames(n, probA, probB)
47        printSummary(winsA, winsB)
48    main()
```

上述代码执行结果如下：

```
>>>
这个程序模拟两个选手 A 和 B 的某种竞技比赛
程序运行需要 A 和 B 的能力值（以 0 到 1 之间的小数表示）
请输入选手 A 的能力值(0-1): 0.45
请输入选手 B 的能力值(0-1): 0.5
模拟比赛的场次: 1000
竞技分析开始，共模拟 1000 场比赛
选手 A 获胜 371 场比赛，占比 37.1%
选手 B 获胜 629 场比赛，占比 62.9%
```

最后再回到体育竞技分析问题，通过模拟方法分析球员之间能力的微小差异带来的比赛结果不同，是否会产生能力差别小却导致比赛结果一边倒的现象？假设 A 在发球局赢得了 45%比赛，而他的对手 B 发球局比他多赢了 5%。结果显示，尽管能力上只有很小的差距（5%），但是 A 大约需要经历 3 场比赛才能赢一场，他赢得一场 3 局或 5 局的比赛机会十分渺茫。进一步地，可以将这个程序扩展为羽毛球、乒乓球、网球等多种模式，可以找到体育竞技规律。当然，深入探讨竞技规律的前提是参赛选手水平差别不大，所发现的规律将有助于弥补短板，类似中国男足与巴西男足的竞技规律是没必要探讨的。

3. 设计过程总结

本结合体育竞技实例介绍了自顶向下的设计过程。从问题输入输出确定开始，

整体设计逐渐向下进行。每一层以大体算法描述开始，然后逐步细化成代码，细节被函数封装，整个过程可以概括为以下 4 个步骤。

步骤 1：将算法表达为一系列小问题。

步骤 2：为每个小问题设计接口。

步骤 3：通过将算法表达为接口关联的多个小问题来细化算法。

步骤 4：为每个小问题重复上述过程。

自顶向下设计是一种开发复杂程序最具价值的设计理念和工具，设计过程自然且简单，自顶向下设计通过封装实现抽象，利用了模块化设计的思想。

8.3.2　自底向上执行

程序编写后，需要经过测试过程。对于较小规模程序，直接运行即可。但对于稍微大规模的程序，需要特殊方法应对测试问题。就像自顶向下设计，每次只设计程序的一部分比一下子解决整个问题更容易，开展测试的更好办法也是将程序分成小部分逐个测试。对于 Python 语言，执行和测试含义相同，本书将交替使用这两个词语，不作区分。

执行中等规模程序的最好方法是从结构图最底层开始，而不是从顶部开始，然后逐步上升。或者说，先运行和测试每一个基本函数，再测试由基础函数组成的整体函数，这样有助于定位错误。在体育竞技分析实例中，参考图 8.3，可以从 gameOver()函数开始测试。Python 解释器提供 import 保留字辅助开展单元测试，语法格式如下：

```
import <源文件名称>
```

这里需要注意，import 要求源文件名称中不能出现英文句号（.），因此，需要对 e15.1MatchAnalysis.py 文件修改名称，这里改为 e151MatchAnalysis.py。之后，可以对 gameOver()函数进行单元测试，代码如下：

```
>>>import e151MatchAnalysis
>>>e151MatchAnalysis.gameOver(15, 10)
True
>>>e151MatchAnalysis.gameOver(10, 1)
False
```

通过输入比赛分数可以测试 gameOver()函数的执行结果，初步测试说明 gameOver()函数是正确的。可以进一步测试 simOneGame()函数，代码如下：

```
>>>import e151MatchAnalysis
>>>e151MatchAnalysis.simOneGame(.45, .5)
(9, 15)
>>>e151MatchAnalysis.simOneGame(.45, .5)
(15, 13)
```

注意到当概率相等时，比分也十分接近。当概率相差很远时，比分则成压倒性趋势。这与该函数的预期结果是相符合的。

通过继续进行这样的单元测试可以检测程序中的每个函数。独立检验每个函数更容易发现错误。通过模块化设计可以分解问题使编写复杂程序成为可能，通过单元测试方法分解问题使运行和调试复杂程序成为可能。自顶向下和自底向上贯穿程序设计和执行的整个过程。

拓展： 软件开发模型

软件开发模型是指软件开发全部过程、活动和任务的结构框架。软件开发包括需求、设计、编码和测试等阶段，有时也包括维护阶段。软件开发模型能清晰、直观地表达软件开发全过程，明确规定了要完成软件的主要活动和任务，用来作为软件项目工作的基础。对于不同的软件系统，可以采用不同的开发方法，使用不同的编程语言，组织不同技能的人员，运用不同的管理方法等。

阶段测试 8-1：程序设计方法小测验



思考与练习

8.6 什么是自顶向下设计？什么是自底向上执行？两者有何关系？

8.7 自顶向下设计的本质是什么？

8.8 下面能支持自顶向下设计方法的是（ ）。

A．对象 B．循环结构 C．函数 D．过程

8.4 模块 7：pyinstaller 库的使用

要点： pyinstaller 是一个将 Python 语言脚本（.py 文件）打包成可执行文件的第三方库，可用于 Windows、Linux、Mac OS X 等操作系统。

8.4.1 pyinstaller 概述

pyinstaller 是一个十分有用的第三方库，它能够在 Windows、Linux、Mac OS X 等操作系统下将 Python 源文件打包，通过对源文件打包，Python 程序可以在没有安装 Python 的环境中运行，也可以作为一个独立文件方便传递和管理。pyinstaller 需要在命令行（控制台）下用 pip 工具安装，代码如下：

```
:\>pip install pyinstaller
```

或

```
:\>pip3 install pyinstaller
```

pyinstaller 的官方网站网址为 http://www.pyinstaller.org/。

pyinstaller 库会自动将 pyinstaller 命令安装到 Python 解释器目录中，与 pip 或 pip3 命令路径相同，因此可以直接使用。使用 pyinstaller 库十分简单，以实例 2 的实例代码 2.3 为例，在 Windows 平台的命令行中输入 Python 源文件名称，可以使用相对路径或绝对路径，代码如下。

请注意，由于 pyinstaller 不支持源文件名中有英文句号（.）存在，请将实例代码 2.3 文件改为 dpython.py，并假设 dpython.py 文件在 D:\codes 目录中。

```
:\>pyinstaller dpython.py
```

或

```
:\>pyinstaller D:\codes\dpython.py
```

执行完毕后，源文件所在目录将生成 dist 和 build 两个文件夹。其中，build 目录是 pyinstaller 存储临时文件的目录，可以安全删除。最终的打包程序在 dist 内部的 dpython 目录中。目录中其他文件是可执行文件 dpython.exe 的动态链接库。

可以通过-F 参数对 Python 源文件生成一个独立的可执行文件，代码如下：

```
:\>pyinstaller -F dpython.py
```

执行后在 dist 目录中出现了 dpython.exe 文件，没有任何依赖库，执行它即可。

使用 pyinstaller 库需要注意以下问题。

（1）文件路径中不能出现空格和英文句号（.）。

（2）源文件必须是 UTF-8 编码，暂不支持其他编码类型。采用 IDLE 编写的源文件都保存为 UTF-8 编码形式，可直接使用。

拓展： 动态链接

动态链接提供了一种方法，能够使进程在运行时实际调用不属于其程序的代码。如果其他代码由操作系统提供，则应用程序由于不包含这些代码而变得十分精简。Windows 平台提供大量的动态链接库，一般使用 dll 或 ocx 为扩展名。

静态链接与动态链接相对，指程序中自包含其所调用的所有代码，这使程序可以在系统间移动而无须考虑库函数是否一致。

8.4.2　pyinstaller 解析

pyinstaller 有一些常用参数，如表 8.1 所示。

表 8.1　pyinstaller 命令的常用参数

参　数	功　能
-h, --help	查看帮助
-v, --version	查看 pyinstaller 版本
--clean	清理打包过程中的临时文件

续表

参　数	功　能
-D, --onedir	默认值，生成 dist 目录
-F, --onefile	在 dist 文件夹中只生成独立的打包文件
-p DIR, --paths DIR	添加 Python 文件使用的第三方库路径
-i <.ico or .exe,ID or .icns> --icon <.ico or .exe,ID or .icns>	指定打包程序使用的图标（icon）文件

pyinstaller 命令不需要在 Python 源文件中增加代码，只需要通过命令行进行打包即可。-F 参数最为常用，对于包含第三方库的源文件，可以使用-p 添加第三方库所在路径。如果第三方库由 pip 安装且在 Python 环境目录中，则不需要使用-p 参数。以实例 10 的实例代码 9.3 为例，该代码使用了 jieba 库，将该文件改名为 caltk.py，打包方法如下：

```
:\>pyinstaller -F D:\codes\caltk.py
```

在 dist 目录中将生成打包文件 caltk.exe，将三国演义.txt 文件复制到 dist 目录中，执行该程序：

```
:\ >D:\codes\dist\caltk.exe
<此处略去程序运行后的结果>
```

思考与练习

8.9　pyinstaller 命令最常使用的参数有哪些？

8.10　如果 Python 源文件使用了第三方库，如何使用 pyinstaller 命令？

8.11　对 Python 源文件打包有哪些优缺点？

8.5　计算生态和模块编程

要点： Python 语言有 9 万多个第三方库，形成了庞大的计算生态，请读者建立模块编程思想。

近 20 年的开源运动产生了深植于各信息技术领域的大量可重用资源，直接且有力地支撑了信息技术超越其他技术领域的发展速度，形成了“计算生态”。产业界广泛利用可重用资源快速构建应用已经是主流产品开发方式。Python 语言从诞生之初致力于开源开放，建立了全球最大的编程计算生态。

Python 官方网站提供了第三方库索引功能（the Python Package Index，PyPI），网址如下：

https://pypi.python.org/pypi

该页面列出了 Python 语言 9 万多个第三方库的基本信息，这些函数库覆盖信息领域

技术所有技术方向。这里需要说明的是，Python 语言的函数库并非都采用 Python 语言编写。由于 Python 有非常简单灵活的编程方式，很多采用 C、C++等语言编写的专业库可以经过简单的接口封装供 Python 语言程序调用。这样的黏性功能使得 Python 语言成为了各类编程语言之间的接口，Python 语言也被称为“胶水语言”。

正是因为 Python 语言有了胶水的黏性，围绕它迅速形成了全球最大的编程语言开放社区，建立了 9 万多个第三方库的庞大规模，构建了计算生态。

30 年前，计算机领域还处于刀耕火种年代，编写程序仅能调用官方提供的 API 功能。20 年前，随着开源运动的兴起和蓬勃发展，一批开源项目诞生，降低了专业人士编写程序的难度，实现了专业级别的代码复用。10 年前，开源运动深入开展，专业人士开始大量贡献各领域最优秀的研究和开发成果，并通过开源库形式发布出来。那今天呢？编程领域形成了庞大的计算生态，需要一种编程语言或方式将不同语言、不同特点、不同使用方式的代码统一起来。历史选择了 Python 语言，Python 语言也证明了它的价值。

Python 第三方程序包括库（library）、模块（module）、类（class）和程序包（Package）等多种命名，本书不对这些命名进行区分，统一将这些可重用代码统称为“库”。Python 内置的库称为标准库，其他库称为第三方库。

在计算生态思想指导下，编写程序的起点不再是探究每个具体算法的逻辑功能和设计，而是尽可能利用第三方库进行代码复用，探究运用库的系统方法。这种像搭积木一样的编程方式，称为“模块编程”。每个模块可能是标准库、第三方库、用户编写的其他程序或对程序运行有帮助的资源等。模块编程与模块化设计不同，模块化设计主张采用自顶向下设计思想，主要开展耦合度低的单一程序设计与开发，而模块编程主张利用开源代码和第三方库作为程序的部分或全部模块，像搭积木一样编写程序。

本书建立在“计算生态”理念基础上，试图通过 10 余个各类函数库和 20 余个实例讲解让读者理解并学会运用计算生态，编写功能更强大的程序。

拓展： Python——构建计算生态的编程语言

——学了编程能做什么？

——你将走进信息时代的大门，能够运用信息时代的最新成果。

学了编程能做什么？似乎只能打印字符、挑战汉诺塔。这是大多数编程语言学习预期和学习效果的鸿沟。因为绝大多数编程语言设计用于开发专业功能，而不是用于构建计算生态，因此，需要专业程序员经过漫长学习才能够掌握并开发有价值的程序。Python 语言却不同，它不是其他语言的替代，而是一种真正面向计算生态的语言。

——AlphaGo 很炫，它打败了世界上最厉害的人类围棋选手。

——AlphaGo 开源了，采用 Python 语言，快去试用看看吧。

思考与练习

8.12 在哪里可以找到 Python 的第三方库列表？

8.13 模块编程是什么含义？

8.14 思考实例 12 的功能，如果不采用第三方库，需要学习哪些专业知识才能写出程序？

8.6 Python 第三方库的安装

要点：Python 第三方库有 3 种安装方式。

Python 语言有标准库和第三方库两类库，标准库随 Python 安装包一起发布，用户可以随时使用，第三方库需要安装后才能使用。由于 Python 语言经历了版本更迭过程，而且，第三方库由全球开发者分布式维护，缺少统一的集中管理，因此，Python 的第三方库曾经一度制约了 Python 语言的普及和发展。随着官方 pip 工具的应用，Python 第三方库的安装变得十分容易。

Python 第三方库依照安装方式灵活性和难易程度有 3 个安装方法，建议读者依次使用，能够将第三方库安装成功，这 3 个方法是 pip 工具安装、自定义安装和文件安装。

8.6.1 pip 工具安装

最常用且最高效的 Python 第三方库安装方式是采用 pip 工具安装。pip 是 Python 官方提供并维护的在线第三方库安装工具。对于同时安装 Python 2 和 Python 3 环境的系统，建议采用 pip3 命令专门为 Python 3 版本安装第三方库。为了叙述方便，本文后续都采用 pip 代替 pip 或 pip3 命令。

pip 是 Python 内置命令，需要通过命令行执行，执行 pip -h 命令将列出 pip 常用的子命令，注意，不要在 IDLE 环境下运行 pip 程序。

```
:\>pip -h
Usage:
  pip <command> [options]

Commands:
  install        Install packages.
  download       Download packages.
  uninstall      Uninstall packages.
  freeze         Output installed packages in requirements format.
  list           List installed packages.
  show           Show information about installed packages.
  search         Search PyPI for packages.
  wheel          Build wheels from your requirements.
```

```
  hash          Compute hashes of package archives.
  completion    A helper command used for command completion
  help          Show help for commands.
```

pip 支持安装（install）、下载（download）、卸载（uninstall）、列表（list）、查看（show）、查找（search）等一系列安装和维护子命令。

安装一个库的命令格式如下：

```
pip install <拟安装库名>
```

例如，安装 pygame 库，pip 工具默认从网络上下载 pygame 库安装文件并自动安装到系统中。

```
:\>pip install pygame
...
Installing collected packages: pygame
Successfully installed pygame-1.9.2b1
```

使用-U 标签可以更新已安装库的版本，例如，用 pip 更新本身：

```
:\>pip install -U pip
Requirement already up-to-date: pip in d:\python35-32\lib\site-packages
```

卸载一个库的命令格式如下：

```
pip uninstall <拟卸载库名>
```

例如，卸载 pygame 库，卸载过程可能需要用户确认。

```
:\>pip uninstall pygame
...
Successfully uninstalled pygame-1.9.2b1
```

可以通过 list 子命令列出当前系统中已经安装的第三方库，例如：

```
pip list
```

执行效果如下，部分结果省略：

```
:\>pip list
beautifulsoup4 (4.5.1)
bottle (0.12.9)
click (6.6)
cycler (0.10.0)
decorator (4.0.10)
Django (1.10.1)
…
```

pip 的 show 子命令列出某个已经安装库的详细信息，例如：

```
pip show <拟查询库名>
```

以 sip 库为例，执行效果如下：

```
:\>pip show sip
---
Metadata-Version: 1.1
Name: sip
Version: 4.18.1
Summary: Python extension module generator for C and C++ libraries
Home-page: https://www.riverbankcomputing.com/software/sip/
Author: Riverbank Computing Limited
Author-email: info@riverbankcomputing.com
Installer: pip
License: None
Location: d:\python35-32\lib\site-packages
Requires:
Classifiers:
```

pip 的 download 子命令可以下载第三方库的安装包，但并不安装，例如：

```
pip download <拟下载库名>
```

以下载 PyQt5 为例，执行效果如下：

```
:\>pip download PyQt5
Collecting sip (from PyQt5)
  Downloading sip-4.18.1-cp35-none-win32.whl
  Saved c:\windows\system32\sip-4.18.1-cp35-none-win32.whl
Successfully downloaded PyQt5 sip
```

pip 的 search 子命令可以联网搜索库名或摘要中的关键字，例如：

```
pip search <拟查询关键字>
```

以查询 installer 为例，执行效果如下：

```
:\>pip search installer
winbrew (1.1.7)           - Native package installer for Windows
pygitflow-avh (1.2.0)     - Pythonic Installer for Git Flow
                            (AVH Edition).
notouch (0.3)             - Notouch Physical Machine
                            Installer Automation Service
...
```

pip 是 Python 第三方库最主要的安装方式，可以安装超过 90%以上的第三方库。然而，由于一些历史、技术和政策等原因，还有一些第三方库暂时无法用 pip 安装，此时，需要其他的安装方法。

pip 工具与操作系统也有关系，在 Mac OS X 和 Linux 等操作系统中，pip 工具几乎可以安装任何 Python 第三方库，在 Windows 操作系统中，有一些第三方库仍然需要用其他方式尝试安装。

8.6.2　自定义安装

自定义安装指按照第三方库提供的步骤和方式安装。第三方库都有主页用于维护库的代码和文档。以科学计算用的 numpy 为例，开发者维护的官方主页如下：

http://www.numpy.org/

浏览该网页找到下载链接，如下：

http://www.scipy.org/scipylib/download.html

进而根据指示步骤安装。

自定义安装一般适合用于 pip 中尚无登记或安装失败的第三方库。

8.6.3　文件安装

由于 Python 某些第三方库仅提供源代码，通过 pip 下载文件后无法在 Windows 系统编译安装，会导致第三方库安装失败。在 Windows 平台下所遇到的无法安装第三方库的问题大多属于这类。

为了解决这类第三方库安装问题，美国加州大学尔湾分校提供了一个页面，帮助 Python 用户获得 Windows 可直接安装的第三方库文件，链接地址如下：

http://www.lfd.uci.edu/~gohlke/pythonlibs/。

该地址列出了一批在 pip 安装中可能出现问题的第三方库。这里以 scipy 为例说明，首先在上述页面中找到 scipy 库对应的内容，如图 8.4 所示。

```
SciPy is software for mathematics, science, and engineering.
   Requires numpy+mkl.
   Install numpy+mkl before installing scipy.
   scipy-0.18.1-cp27-cp27m-win32.whl
   scipy-0.18.1-cp27-cp27m-win_amd64.whl
   scipy-0.18.1-cp34-cp34m-win32.whl
   scipy-0.18.1-cp34-cp34m-win_amd64.whl
   scipy-0.18.1-cp35-cp35m-win32.whl
   scipy-0.18.1-cp35-cp35m-win_amd64.whl
```

图 8.4　scipy 库对应的 Windows 文件下载页面

选择其中的.whl 文件下载，这里选择适用于 Python 3.5 版本解释器和 32 位系统的对应文件：scipy-0.18.1-cp35-cp35m-win32.whl，下载该文件到 D:\pycodes 目录。然后，采用 pip 命令安装该文件。

```
:\>pip install D:\pycodes\scipy-0.18.1-cp35-cp35m-win32.whl
Processing d:\pycodes\scipy-0.18.1-cp35-cp35m-win32.whl
Installing collected packages: scipy
Successfully installed scipy-0.18.1
```

拓展： whl 格式

whl 是 Python 库的一种打包格式，用于通过 pip 进行安装，相当于 Python 库的安装包文件。whl 文件本质上是一个压缩格式文件，可以通过改扩展名为 zip 查看其中内容。whl 格式用于替代 Python 早期的 eggs 格式，是 Python 打包格式的事实标准。

对于上述 3 种安装方式，一般优先选择采用 pip 工具安装，如果安装失败，则选择自定义安装或者文件安装（Windows 平台）。另外，如果需要在没有网络条件下安装 Python 第三方库，请直接采用文件安装方式。其中，.whl 文件可以通过 pip download 指令在有网络条件的情况下获得。

思考与练习

8.15 pip 工具最常用的子命令是什么？

8.16 如果 pip 工具无法安装第三方库，还有哪些其他办法？

8.17 下载一些 Python 第三方库文件（.whl 文件），尝试通过文件方式安装。

8.7 实例 16：pip 安装脚本

要点： 这是一个用 pip 安装第三方库的例子。

Python 安装包自带工具 pip（或 pip3）是安装第三方库最重要的方法。pip 的使用十分方便，本节介绍一些重要的第三方库，同时请读者用 pip 安装这些函数库。

实例 16 共需要安装 20 个第三方 Python 库，如表 8.2 所示，需要注意的是，库名是第三方库常用的名字，pip 安装用的名字和库名不一定完全相同，建议采用小写字符。

表 8.2 第三方 Python 库（共 20 个）

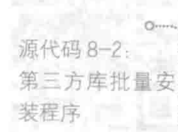
库　　名	用　　途	pip 安装指令
NumPy	矩阵运算	pip install numpy
Matplotlib	产品级 2D 图形绘制	pip install matplotlib
PIL	图像处理	pip install pillow
sklearn	机器学习和数据挖掘	pip install sklearn
Requests	HTTP 协议访问	pip install requests
Jieba	中文分词	pip install jieba
Beautiful Soup 或 bs4	HTML 和 XML 解析	pip install beautifulsoup4
Wheel	Python 文件打包	pip install wheel
pyinstaller	打包 Python 源文件为可执行文件	pip install pyinstaller
Django	Python 最流行的 Web 开发框架	pip install django
Flask	轻量级 Web 开发框架	pip install flask

续表

库　　名	用　　途	pip 安装指令
WeRoBot	微信机器人开发框架	pip install werobot
Networkx	复杂网络和图结构的建模和分析	pip install networkx
SymPy	数学符号计算	pip install sympy
pandas	高效数据分析	pip install pandas
PyQt5	基于 Qt 的专业级 GUI 开发框架	pip install pyqt5
PyOpenGL	多平台 OpenGL 开发接口	pip install pyopengl
PyPDF2	PDF 文件内容提取及处理	pip install pypdf2
docopt	Python 命令行解析	pip install docopt
PyGame	简单小游戏开发框架	pip install pygame

安装过程请在系统命令行下进行，而不要在 IDLE 中，部分库会依赖其他函数库，pip 会自动安装，部分库下载后需要一个安装过程，pip 也会自动执行。成功安装库后会出现“Successfully installed…”提示，效果如下：

```
:\>pip install pygame
...
Installing collected packages: pygame
Successfully installed pygame-1.9.2b1
```



源代码 8-2：第三方库批量安装程序



如果读者希望自动安装这些库，可以使用 Python 标准库 os 的 system()函数调用控制台。实例代码 16.1 给出了采用 pip 批量安装 Python 库的方法。

实例代码 16.1　　e16.1BatchInstall.py

```
1  #e16.1BatchInstall.py
2  import os
3  libs = {"numpy","matplotlib","pillow","sklearn","requests",\
          "jieba","beautifulsoup4","wheel","networkx","sympy",\
          "pyinstaller","django","flask","werobot","PyQt5",\
          "pandas","pyopengl","pypdf2","docopt","pygame"}
4  try:
5      for lib in libs:
6          os.system("pip install "+lib)
7      print("Successful")
8  except:
9      print("Failed Somehow")
```

拓展： PyPI 的权重值

PyPI 提供了第三方库的索引，除了基本信息外，PyPI 还根据每个库被检索和下载的情况计算了权重值（Weight）。由于第三方库的开发没有任何规划，对于某个功能将有一批库可以支持，权重值较高的库往往质量更好。

思考与练习

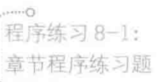
8.18 什么情况下使用 pip3 指令安装第三方库？

8.19 os.system()函数的功能是什么？

8.20 请读者调研选取一个最感兴趣的第三方库，并将它安装在系统中。

本章小结

本章阐述了计算思维的概念，以体育竞技分析为例介绍了自顶向下的设计方法和自底向上的测试方法。进一步阐述了利用 Python 第三方库编程的模块编程思想和计算生态的理解和运用。








程序练习题



8.1 借鉴实例 15 的思路，采用乒乓球规则模拟比赛，分析体育竞技规律。

8.2 借鉴实例 15 的思路，采用篮球规则模拟比赛，分析体育竞技规律。比较结果与程序练习题 8.1 的不同。

8.3 You-Get 是一个基于 Python 3 的视频下载工具，支持多数国内外主流视频站点的视频下载。请查找该项目的主页，安装这个第三方库并编写一个实例。

8.4 词云是设计和统计的结合，也是艺术与计算机科学的碰撞。Wordcloud 是一款基于 Python 的词云第三方库，支持对词语数量、背景蒙版、字体颜色等各种细节的设置，试结合 jieba 的分词功能构建《三国演义》的词云效果。

第9章　科学计算和可视化

电子教案 9-1：科学计算和可视化



计算不再关乎计算机，它与生活处处相关。

Computing is not about computers any more. It is about living.

——尼古拉斯·尼葛洛庞帝（Nicholas Negroponte）

麻省理工学院媒体实验室的创办人

学习目标

（1）了解科学计算的基本概念。

（2）了解数据可视化的概念。

（3）运用科学计算库进行矩阵分析和数值运算。

（4）了解图像的矩阵表示和处理。

（5）运用数据绘图库进行坐标系绘制。

（6）运用数据绘图库进行雷达图绘制。

照片照得好,不如滤镜用得好！一款好的滤镜软件可以让照片呈现不一样的风格乃至风情，修理照片需要扬长避短达到最佳效果。可是滤镜款式千百种，却没有一款专门为你设计？不如自己来写个滤镜吧。

本章将以 PIL 库和 numpy 库为基础零起点编写一款手绘风的高逼格滤镜。

9.1 问题概述

> **要点**：科学计算需要采用矩阵运算库 numpy 和绘制库 matplotlib。

人类认识世界遵循由表及里、由定性到定量、由数据到规律的过程。无论是说明事物属性、展示数据规律、阐述规律原理，还是论述观点、支持决策、预测分析，都离不开基于数学和运算的科学表达，这需要科学计算的支持。科学计算是为了解决科学和工程中数学问题而利用计算机进行的数值计算，它不仅是科学家在运算自然规律时所采用的方法，更是普通人提升专业化程度的必要手段。Python 语言为开展人人都能使用的科学计算提供了有力支持。

开展基本的科学计算需要两个步骤：组织数据和展示数据。组织数据是运算的基础，也是将客观世界数字化的必要手段；展示数据是体现运算结果的重要方式，也是展示结论的有力武器。本章将分别介绍用于组织和运算数据的第三方 Python 库 numpy 和展示数据并绘制专业图表的第三方库 matplotlib。

——不管怎么说，科学计算都是科学家该做的事，与普通人无关。

——想不想用程序生成照片的手绘效果？

请读者快速浏览 9.3 节、9.5 节和 9.6 节，这些实例说明，科学计算不仅能够展示数字结果，还能够与 PIL 图像库混合使用产生有趣的手绘效果，更能够让数据展示变得非常专业。掌握这些能力，仅需要了解 Python 的两个第三方库。

先补充一个简单的数学概念——矩阵。数学的矩阵（Matrix）是一个按照长方阵列排列的复数或实数集合，最早来自于方程组的系数及常数所构成的方阵。矩阵是高等代数学中的常见工具，主要应用于统计数学、物理学、电路学、力学、光学、量子物理、计算机图像和动画等领域。

传统的科学计算主要基于矩阵运算，因为大量数值通过矩阵可以有效组织和表达。科学计算领域最著名的计算平台 Matlab 采用矩阵作为最基础的变量类型。矩阵有维度概念，一维矩阵是线性的，类似于列表，二维矩阵是表格状的，这是常用的数据表示形式。科学计算与传统计算的一个显著区别在于，科学计算以矩阵而不是单一数值为基础，增加了计算密度，能够表达更为复杂的数据运算逻辑。

拓展：离散和连续

> 矩阵是一个典型的离散变量类型，它将一些数据组织到一起。世界是连续的还是离散的呢？从人类观测角度，世界可以被解释成一个个离散的观测值；从微观角度，世界是原子不停运动的结果，应该是连续的；再微观到量子力学角度，任何连续运动都是最小粒子量子运动的结果，世界应该是离散的。宇宙真有最小粒子吗？
>
> ——世界是不确定的，还是确定的？世界是概率的，还是微积分的？
>
> ——醒醒，开始看程序！

思考与练习

9.1　请思考在日常工作和生活中科学计算还有什么应用。

9.2　请尝试安装 numpy 和 matplotlib 库。

9.2　模块8：numpy库的使用

> 要点：numpy是用于处理含有同种元素的多维数组运算的第三方库。

9.2.1　numpy库概述

图片资料9-1：Python 快速参考之 numpy 库



Python 标准库中提供了一个 array 类型，用于保存数组类型数据，然而这个类型不支持多维数据，处理函数也不够丰富，不适合数值运算。因此，Python 语言的第三方库 numpy 得到了迅速发展，至今，numpy 已经成为了科学计算事实上的标准库。

numpy 库处理的最基础数据类型是由同种元素构成的多维数组（ndarray），简称“数组”。数组中所有元素的类型必须相同，数组中元素可以用整数索引，序号从 0 开始。ndarray 类型的维度（dimensions）叫做轴（axes），轴的个数叫做秩（rank）。一维数组的秩为 1，二维数组的秩为 2，二维数组相当于由两个一维数组构成。

由于 numpy 库中函数较多且命名容易与常用命名混淆，建议采用如下方式引用 numpy 库：

```
>>>import numpy as np
```

其中，as 保留字与 import 一起使用能够改变后续代码中库的命名空间，有助于提高代码可读性。简单地说，在程序的后续部分中，np 代替 numpy。

9.2.2　numpy库解析

numpy 库常用的创建数组（ndarray 类型）函数共有 7 个，如表 9.1 所示。

表 9.1　numpy 库常用的数组创建函数（共 7 个）

函　数	描　述
np.array([x,y,z], dtype=int)	从 Python 列表和元组创造数组
np.arange(x,y,i)	创建一个由 x 到 y，以 i 为步长的数组
np.linspace(x,y,n)	创建一个由 x 到 y，等分成 n 个元素的数组
np.indices((m,n))	创建一个 m 行 n 列的矩阵
np.random.rand(m,n)	创建一个 m 行 n 列的随机数组

续表

函　　数	描　　述
np.ones((m,n),dtype)	创建一个 m 行 n 列全 1 的数组，dtype 是数据类型
np.empty((m,n),dtype)	创建一个 m 行 n 列全 0 的数组，dtype 是数据类型

创建一个简单的数组后，可以查看 ndarray 类的基本属性，如表 9.2 所示。

表 9.2　ndarray 类的常用属性（共 7 个）

属　　性	描　　述
ndarray.ndim	数组轴的个数，也被称作秩
ndarray.shape	数组在每个维度上大小的整数元组
ndarray.size	数组元素的总个数
ndarray.dtype	数组元素的数据类型，dtype 类型可以用于创建数组
ndarray.itemsize	数组中每个元素的字节大小
ndarray.data	包含实际数组元素的缓冲区地址
ndarray.flat	数组元素的迭代器

使用实例如下：

```
>>>import numpy as np
>>>a = np.ones((4,5))
>>>print(a)
[[ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]]
>>>a.ndim
2
>>>a.shape
(4,5)
>>>a.dtype
dtype('float64')
```

数组在 numpy 中被当作对象，可以采用<a>.<b>()方式调用一些方法。表 9.3 给出了改变数组基础形态的操作方法，例如改变和调换数组维度等。其中，np.flatten()函数用于数组降维，相当于平铺数组中的数据，该功能在矩阵运算及图像处理中用处很大。

表 9.3　ndarray 类的形态操作方法（共 5 个）

方　　法	描　　述
ndarray.reshape(n,m)	不改变数组 ndarray，返回一个维度为(n,m)的数组
ndarray.resize(new_shape)	与 reshape()作用相同，直接修改数组 ndarray
ndarray.swapaxes(ax1, ax2)	将数组 n 个维度中任意两个维度进行调换
ndarray. flatten()	对数组进行降维，返回一个折叠后的一维数组
ndarray.ravel()	作用同 np.flatten()，但是返回数组的一个视图

表 9.4 给出了 ndarray 类的索引和切片方法。数组切片得到的是原始数组的视图，所有修改都会直接反映到源数组。如果需要得到 ndarray 切片的一份副本，需要进行复制操作，比如 arange[5:8].copy()。

表 9.4　ndarray 类的索引和切片方法（共 5 个）

方　法	描　述
x[*i*]	索引第 *i* 个元素
x[-*i*]	从后向前索引第 *i* 个元素
x[*n*:*m*]	默认步长为 1，从前往后索引，不包含 *m*
x[-*m*:-*n*]	默认步长为 1，从后往前索引，结束位置为 *n*
x[*n*,*m*,*i*]	指定 *i* 步长的由 *n* 到 *m* 的索引

使用实例如下：

```
>>>a = np.random.rand(5,3) #生成 5×3 的数组，用随机数填充
>>>a[2]  #获得第 2 行数据
array([ 0.78426574,  0.60171943,  0.98825306])
>>>a[1:3]
array([[ 0.49276756,  0.44735929,  0.10356773],
      [ 0.78426574,  0.60171943,  0.98825306]])
>>>a[-5:-2:2]
array([[ 0.95517757,  0.3634953 ,  0.34138831],
      [ 0.78426574,  0.60171943,  0.98825306]])
```

除了 ndarray 类型方法外，numpy 库提供了一批运算函数。表 9.5 列出了 numpy 库的算术运算函数，共 8 个。这些函数中，输出参数 y 可选，如果没有指定，将创建并返回一个新的数组保存计算结果；如果指定参数，则将结果保存到参数中。例如，两个数组相加可以简单地写为 a+b，而 np.add(a,b,a)则表示 a+=b。

表 9.5　numpy 库的算术运算函数（共 8 个）

函　数	描　述
np.add(x1, x2 [, y])	y = x1 + x2
np.subtract(x1, x2 [, y])	y = x1 – x2
np.multiply(x1, x2 [, y])	y = x1 * x2
np.divide(x1, x2 [, y])	y = x1 / x2
np floor_divide(x1, x2 [, y])	y = x1 // x2，返回值取整
np.negative(x [,y])	y = –x
np.power(x1, x2 [, y])	y = x1**x2
np.remainder(x1, x2 [, y])	y = x1 % x2

表 9.6 列出了 numpy 库的比较运算函数，共 7 个。

表 9.6　numpy 库的比较运算函数（共 7 个）

函　数	符号描述
np. equal(x1, x2 [, y])	y = x1 == x2
np. not_equal(x1, x2 [, y])	y = x1 != x2

续表

函　　数	符 号 描 述
np. less(x1, x2, [, y])	y = x1 < x2
np. less_equal(x1, x2, [, y])	y = x1 <= x2
np. greater(x1, x2, [, y])	y = x1 > x2
np. greater_equal(x1, x2, [, y])	y = x1 >= x2
np.where(condition[x,y])	根据给出的条件判断输出 x 还是 y

表 9.6 将返回一个布尔数组，它包含两个数组中对应元素值的比较结果，例子如下。where()函数是三元表达式 x if condition else y 的矢量版本。

```
>>>np.less([1,2],[2,2])
array([ True, False], dtype=bool)
```

numpy 还有其他一些有趣而操作方便的函数，如表 9.7 所示。

表 9.7　numpy 库的其他运算函数（共 9 个）

函　　数	描　　述
np.abs(x)	计算基于元素的整型、浮点或复数的绝对值
np.sqrt(x)	计算每个元素的平方根
np.squre(x)	计算每个元素的平方
np.sign(x)	计算每个元素的符号：1(+)、0、–1(–)
np.ceil(x)	计算大于或等于每个元素的最小值
np.floor(x)	计算小于或等于每个元素的最大值
np.rint (x[, out])	圆整，取每个元素为最近的整数，保留数据类型
np.exp(x[, out])	计算每个元素的指数值
np.log(x), np.log10(x), np.log2(x)	计算自然对数(e)，基于 10、2 的对数，log(1 + x)

numpy 库还包括三角运算函数、傅里叶变换、随机和概率分布、基本数值统计、位运算、矩阵运算等非常丰富的功能，读者在使用时可以到官方网站查询。

拓展： 运算规则

实数的算术运算是最为常见的运算规则，类似的，矩阵也有算术运算。一个完备的运算体系包括运算基本单位和运算规则。在 numpy 中，运算基本单位是数组，运算规则与实数一样，包括算术运算、比较运算、统计运算、三角运算、随机运算等。numpy 库的广泛使用与完备的运算体系密切相关。

思考与练习

9.3　创建一个 ndarray 变量有哪些方法？

9.4　如何对 ndarray 的每个变量求平方根？

9.5　思考 ndarray 的降维是什么含义。

9.3　实例 17：图像的手绘效果

要点：这是一个使用 numpy 和 PIL 库提取图像特征形成手绘效果的实例。

7.2 节使用 PIL 库获取了图像的轮廓，虽然提取了轮廓，但这个轮廓缺少立体感，视觉效果不够丰满。光线照射使立体物出现明暗变化，运用这个原理是空间素描的基本方法，本节介绍采用 Python 程序增加深浅层次变化，从而使图像轮廓更富立体感、空间感和色泽感，接近人类手绘效果。

9.3.1　图像的数组表示

图像是有规则的二维数据，可以用 numpy 库将图像转换成数组对象，以北京故宫的照片为例，名称为 fcity.jpg，放置在 D:\pycodes 目录下，方法如下：

```
>>>from PIL import Image
>>>import numpy as np
>>>im = np.array(Image.open('D:\\pycodes\\fcity.jpg'))
>>>print(im.shape, im.dtype)
(881, 1266, 3) uint8
```

图像转换对应的 ndarray 类型是三维数据，如(881, 1266, 3)，其中，前两维表示图像的长度和宽度，单位是像素，第三维表示每个像素点的 RGB 值，每个 RGB 值是一个单字节整数。

PIL 库包括图像转换函数，能够改变图像单个像素的表示形式。使用 convert() 函数，这是'L'模式，表示将像素从 RGB 的 3 字节形式转变为单一数值形式，这个数值范围为 0～255，表示灰度色彩变化。此时，图像从彩色变为带有灰度的黑白色。转换后，图像的 ndarray 类型变为二维数据，每个像素点色彩只由一个整数表示。

```
>>>im = np.array(Image.open('D:\\pycodes\\fcity.jpg').convert('L'))
>>>print(im.shape, im.dtype)
(881, 1266) uint8
```

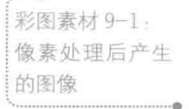
通过对图像的数组转换，可以利用 numpy 访问图像上的任意像素值，例如，获取位于坐标(20, 300)像素的颜色值或获取图像中最大和最小的像素值。也可以采用切片方式获取指定行或列的元素值，甚至修改这些值。

```
>>>print(im[20,300])
120
>>>print(int(im.min()),int(im.max()))
```

```
0 255
>>>print(im[10,:])
[114 113 115 ..., 123 122 123]
```

将图像读入 ndarray 数组对象后，可以通过任意数学操作来获取相应的图像变换。以灰度变换为例，分别对灰度变化后的图像进行反变换、区间变化和像素值平方处理。需要注意的是，有些数学变换会改变图像的数据类型，如变成整数类型等，所以在重新生成 PIL 图像前要先将数据类型通过 numpy.uint()变换成整数，实例如下。像素处理后产生的图像如图 9.1 所示。

```
>>>im0 = np.array(Image.open('np.jpg').convert('L'))
>>>im1 = 255 - im0  #反变换
>>>im2 = (100/255)*im0 + 150  #区间变换
>>>im3 = 255*(im1/255)**2 #像素平方处理
>>>pil_im = Image.fromarray(np.uint(im1)) #分别对 im1、im2、im3 执行
>>>pil_im.show()
```


彩图素材 9-1：像素处理后产生的图像




(a) 原图像


(b) 对 im1 执行的结果


(c) 对 im2 执行的结果


(d) 对 im3 执行的结果

图 9.1 像素处理后产生的图像

拓展： 灰度值

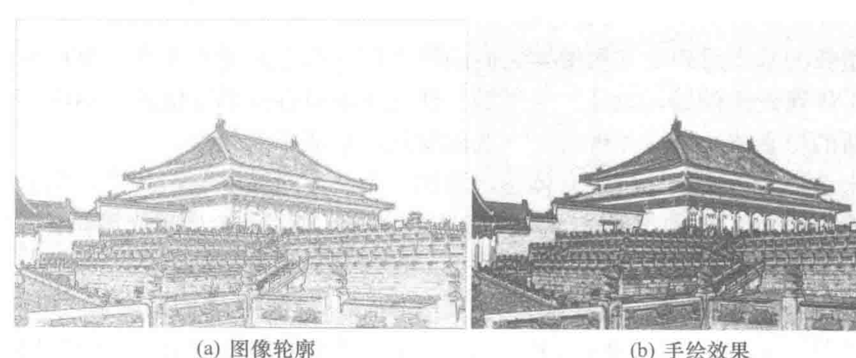
灰度值指黑白图像中点的颜色深度，范围从 0 到 255，黑色为 0，白色为 255，因此，黑白图像也被称为灰度图像。黑白图像主要用于构建非可见光图像，例如医学中超声波形成的图像等。RGB 彩色图片可以通过如下公式转换成灰度值：

$$Gray=R\times0.3+G\times0.59+B\times0.11$$

严格地说，黑白图像是计算机计算能力或存储能力不充分时形成图像的重要方式，如果单个像素点能获得数据值种类超过 256 且计算资源足够，采用彩色图像也可以构建非可见光图谱，例如医学应用中新发展的彩色超声波成像等。

9.3.2　图像的手绘效果

7.2 节介绍了 10 种 ImageFilter 类型的滤镜方法。获得铅笔画风格图像通常采用 ImageFilter.CONTOUR 滤镜，它能够将图像的轮廓信息提取出来，如图 9.2（a）所示。原图像在视觉上更加立体，获得的轮廓图像缺乏立体感。图 9.2（b）给出了希望由程序给出的手绘效果。这个效果用 7.2 节 10 种滤镜都无法实现，该怎么用程序实现呢？



(a) 图像轮廓　　(b) 手绘效果

图 9.2　图像轮廓和手绘效果的对比

彩图素材 9-2：图像轮廓和手绘效果的对比



为了实现手绘风格，即黑白轮廓描绘，首先需要读取原图像的明暗变化，即灰度值。从直观视觉感受上定义，图像灰度值显著变化的地方就是梯度，它描述了图像灰度变化的强度。通常可以使用梯度计算来提取图像轮廓，numpy 中提供了直接获取灰度图像梯度的函数 gradient()，传入图像数组表示即可返回代表 x 和 y 各自方向上梯度变化的二维元组。实例代码 17.1 给出了图像手绘效果的全部代码。

源代码 9-1：图像的手绘效果



实例代码 17.1　　e17.1HandDrawPic.py

```
1  #e19.1HandDrawPic.py
2  from PIL import Image
3  import numpy as np
4  vec_el = np.pi/2.2         # 光源的俯视角度，弧度值
5  vec_az = np.pi/4.          # 光源的方位角度，弧度值
6  depth = 10.                # (0-100)
7  im = Image.open('fcity.jpg').convert('L')
8  a = np.asarray(im).astype('float')
9  grad = np.gradient(a)   #取图像灰度的梯度值
```

```
10  grad_x, grad_y = grad                        #分别取横纵图像梯度值
11  grad_x = grad_x*depth/100.
12  grad_y = grad_y*depth/100.
13  dx = np.cos(vec_el)*np.cos(vec_az)          #光源对 x 轴的影响
14  dy = np.cos(vec_el)*np.sin(vec_az)          #光源对 y 轴的影响
15  dz = np.sin(vec_el)                          #光源对 z 轴的影响
16  A = np.sqrt(grad_x**2 + grad_y**2 + 1.)
17  uni_x = grad_x/A
18  uni_y = grad_y/A
19  uni_z = 1./A
20  a2 = 255*(dx*uni_x + dy*uni_y + dz*uni_z) #光源归一化
21  a2 = a2.clip(0,255)
22  im2 = Image.fromarray(a2.astype('uint8')) #重构图像
23  im2.save('fcityHandDraw.jpg')
```

手绘图像的基本思想是利用像素之间的梯度值（而不是像素本身）重构每个像素值。为了体现光照效果，设计一个光源，建立光源对各点梯度值的影响函数，进而运算出新的像素值，从而体现边界点灰度变化，形成手绘效果。

具体来说，为了更好地体现立体感，增加一个 *z* 方向梯度值，并给 *x* 和 *y* 方向梯度值赋权值 depth。这种坐标空间变化相当于给物体加上一个虚拟光源，根据灰度值大小模拟各部分相对于人视角的远近程度，使画面显得有“深度”。

在利用梯度重构图像时，对应不同梯度取 0～255 之间不同的灰度值，depth 的作用在于调节这个对应关系。depth 较小时，背景区域接近白色，画面显示轮廓描绘；depth 较大时，整体画面灰度值较深，近似于浮雕效果。

将光源定义为 3 个参数：方位角 vec_az、俯视角 vec_el 和深度权值 depth。两个角度的设定和单位向量构成了基础的柱坐标系，体现物体相对于虚拟光源的位置，如实例代码 17.1 的第 4 到第 6 行。

通过 np.gradient()函数计算图像梯度值作为新色彩计算的基础。为了更直观地进行计算，可以把角度对应的柱坐标转化为 *xyz* 立体坐标系。d*x*、d*y*、d*z* 是像素点在施加模拟光源后在 *x*、*y*、*z* 方向上明暗度变化的加权向量，如代码第 13 到第 15 行。

A 是梯度幅值，也是梯度大小。各个方向上总梯度除以幅值得到每个像素单元的梯度值。利用每个单元的梯度值和方向加权向量合成灰度值，clip 函数用于预防溢出，并归一化到 0～255 区间。最后从数组中恢复图像并保存。

由于手绘图像算法涉及三维空间映射，请读者尽量理解，并通过修改其中参数获得更多效果。

思考与练习

9.6 numpy 的 ndarray 类型表示的彩色图像是几维？

9.7 如何将彩色图片转换成灰度图片？之后如何处理每一个像素？

9.8 哪个函数能将 numpy 的 ndarray 类型变成图像？

9.4　模块 9：matplotlib 库的使用

要点：matplotlib 是提供数据绘图功能的第三方库，其 pyplot 子库主要用于实现各种数据展示图形的绘制。

9.4.1　matplotlib.pyplot 库概述

matplotlib.pyplot 是 matplotlib 的字库，引用方式如下：

```
>>>import matplotlib.pyplot as plt
```

图片资料 9-2：Python 快速参考之 matplotlib 库



上述语句与 import matplotlib.pyplot 一致，as 保留字与 import 一起使用能够改变后续代码中库的命名空间，有助于提高代码可读性。简单地说，在后续程序中，plt 将代替 matplotlib.pyplot。

为了正确显示中文字体，请用以下代码更改默认设置,其中'SimHei'表示黑体字。

```
>>>import matplotlib
>>>matplotlib.rcParams['font.family']='SimHei'
>>>matplotlib.rcParams['font.sans-serif'] = ['SimHei']
```

拓展：字体

字体是计算机显示字符的方式，均由人工设计，并采用字体库方式部署在计算机中。西文和中文字体都有很多种类，表 9.8 给出最常用的 10 种中文字体及其英文表示，这些字体的英文表示在程序设计中十分常用，但需要注意，部分字体无法在 matplotlib 库中使用。

表 9.8　字体名称的中英文对照

字 体 名 称	字体英文表示
宋体	SimSun
黑体	SimHei
楷体	KaiTi
微软雅黑	Microsoft YaHei
隶书	LiSu
仿宋	FangSong
幼圆	YouYuan
华文宋体	STSong
华文黑体	STHeiti
苹果丽中黑	Apple LiGothic Medium

matplotlib 库由一系列有组织有隶属关系的对象构成，这对于基础绘图操作来说显得过于复杂。因此，matplotlib 提供了一套快捷命令式的绘图接口函数，即 pyplot 子模块。pyplot 将绘图所需要的对象构建过程封装在函数中，对用户提供了更加友好的接口。pyplot 模块提供一批预定义的绘图函数，大多数函数可以从函数名辨别它的功能。

9.4.2 matplotlib.pyplot 库解析

从本节开始，使用 plt 代替 matplotlib.pyplot。plt 子库提供了一批操作和绘图函数，每个函数代表对图像进行的一个操作，比如创建绘图区域、添加标注或者修改坐标轴等。这些函数采用 plt.<b>()形式调用，其中<b>是具体函数名称。

plt 子库中包含了 4 个与绘图区域有关的函数，如表 9.9 所示。

表 9.9 plt 库的绘图区域函数（共 4 个）

函　数	描　述
plt.figure(figsize=None, facecolor=None)	创建一个全局绘图区域
plt.axes(rect, axisbg='w')	创建一个坐标系风格的子绘图区域
plt.subplot(nrows, ncols, plot_number)	在全局绘图区域中创建一个子绘图区域
plt.subplots_adjust()	调整子绘图区域的布局

使用 figure()函数创建一个全局绘图区域，并且使它成为当前的绘图对象，figsize 参数可以指定绘图区域的宽度和高度，单位为英寸。鉴于 figure()函数参数较多，这里采用指定参数名称的方式输入参数。

```
>>> plt.figure(figsize=(8,4))
```

绘制图像之前也可不调用 figure()函数创建全局绘图区域，此时，plt 子库会自动创建一个默认的绘图区域。显示绘图区域的代码如下：

```
>>>plt.figure(figsize=(8,4))
>>>plt.show()
```

subplot()用于在全局绘图区域内创建子绘图区域，其参数表示将全局绘图区域分成 nrows 行和 ncols 列，并根据先行后列的计数方式在 plot_number 位置生成一个坐标系，实例代码如下，3 个参数关系如图 9.3 所示。其中，全局绘图区域被分割成 3×2 的网格，其中，在第 4 个位置绘制了一个坐标系。

```
>>>plt.subplot(324)
>>>plt.show()
```

axes()默认创建一个 subplot(111)坐标系，参数 rec = [left,bottom,width,height]中 4 个变量的范围都为[0,1]，表示坐标系与全局绘图区域的关系；axisbg 指背景色，默认为 white。

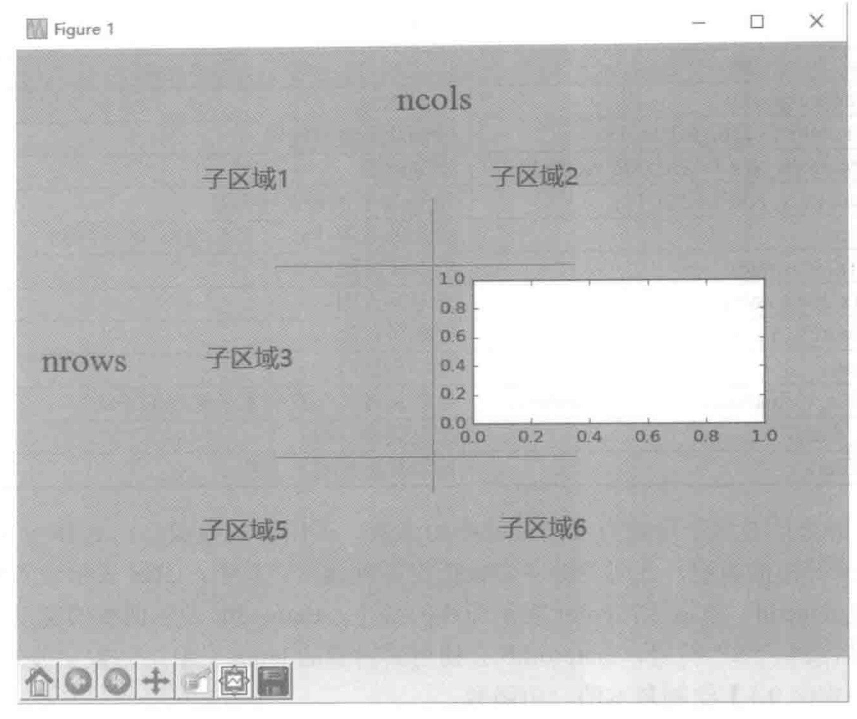


图 9.3　subplot()函数的参数关系

```
>>> plt.axes([0.1, 0.1, 0.7, 0.3], axisbg='y')
>>> plt.show()
```

plt 子库提供了一组读取和显示相关的函数，用于在绘图区域中增加显示内容及读入数据，如表 9.10 所示，这些函数需要与其他函数搭配使用，此处读者有所了解即可。

表 9.10　plt 库的读取和显示函数（共 6 个）

函　数	描　述
plt.legend()	在绘图区域中放置绘图标签（也称图注）
plt.show()	显示创建的绘图对象
plt.matshow()	在窗口显示数组矩阵
plt.imshow()	在 axes 上显示图像
plt.imsave()	保存数组为图像文件
plt.imread()	从图像文件中读取数组

pyplot 模块提供了 17 个用于绘制“基础图表”的常用函数，如表 9.11 所示。

表 9.11　plt 库的基础图表函数（共 17 个）

操　作	描　述
plt.polt(x, y, label, color, width)	根据 x、y 数组绘制直、曲线
plt.boxplot(data, notch, position)	绘制一个箱型图（Box-plot）
plt.bar(left, height, width, bottom)	绘制一个条形图
plt.barh(bottom, width, height, left)	绘制一个横向条形图
plt.polar(theta, r)	绘制极坐标图

续表

操　　作	描　　述
plt.pie(data,explode)	绘制饼图
plt.psd(x, NFFT=256, pad_to, Fs)	绘制功率谱密度图
plt.specgram(x, NFFT=256, pad_to, F)	绘制谱图
plt.cohere (x, y, NFFT=256, Fs)	绘制 X-Y 的相关性函数
plt.scatter()	绘制散点图（x、y 是长度相同的序列）
plt.step(x, y, where)	绘制步阶图
plt.hist(x, bins, normed)	绘制直方图
plt.contour(X, Y, Z, N)	绘制等值线
plt.vlines()	绘制垂直线
plt.stem(x, y, linefmt, markerfmt, basefmt)	绘制曲线每个点到水平轴线的垂线
plt.plot_date()	绘制数据日期
plt.plotfile()	绘制数据后写入文件

plot()函数是用于绘制直线的最基础的函数，调用方式很灵活，x 和 y 可以是 numpy 计算出的数组，并用关键字参数指定各种属性。其中，label 表示设置标签并在图例（legend）中显示，color 表示曲线的颜色，linewidth 表示曲线的宽度。在字符串前后添加“$”符号，matplotlib 会使用其内置的 latex 引擎绘制数学公式。

【微实例 9.1】绘制基本的三角函数。

在坐标系中绘制基本的三角函数，代码如下，绘制效果如图 9.4 所示。

源代码 9-2：基本的三角函数绘制

微实例 9.1　　m9.1PlotTriangle.py

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  x = np.linspace(0, 6, 100)
4  y = np.cos(2 * np.pi * x) * np.exp(-x)+0.8
5  plt.plot(x, y, 'k', color='r', linewidth=3, linestyle="-")
6  plt.show()
```

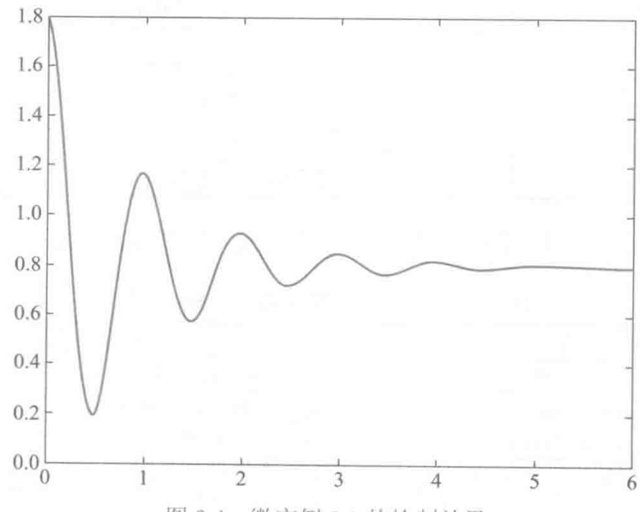


图 9.4　微实例 9.1 的绘制效果

plt 库有两个坐标体系：图像坐标和数据坐标。图像坐标将图像所在区域左下角视为原点，将 x 方向和 y 方向长度设定为 1。整体绘图区域有一个图像坐标，每个 axes()和 subplot()函数产生的子图也有属于自己的图像坐标。axes()函数参数 rect 指当前产生的子区域相对于整个绘图区域的图像坐标。数据坐标以当前绘图区域的坐标轴为参考，显示每个数据点的相对位置，这与坐标系里面标记数据点一致。

表 9.12 给出了与 plt 库的坐标轴设置相关的函数，实例如下。

表 9.12 plt 库的坐标轴设置函数（共 9 个）

函　　数	描　　述
plt.axis('v','off','equal','scaled','tight','image')	获取设置轴属性的快捷方法
plt.xlim(xmin, xmax)	设置当前 x 轴取值范围
plt.ylim(ymin,ymax)	设置当前 y 轴取值范围
plt.xscale()	设置 x 轴缩放
plt.yscale()	设置 y 轴缩放
plt.autoscale()	自动缩放轴视图的数据
plt.text(x,y,s,fontdic,withdash)	为 axes 图轴添加注释
plt.thetagrids(angles, labels, fmt, frac)	设置极坐标网格 theta 的位置
plt.grid(on/off)	打开或者关闭坐标网格

```
>>>plt.plot([1,2,4], [1,2,3])
>>>plt.axis() #获得当前坐标轴范围
(1.0, 4.0, 1.0, 3.0)
>>>plt.axis([0,5,0,8]) #4 个变量分别是[xmin,xmax,ymin,ymax]
>>>plt.show() #请读者观察输出结果
```

表 9.13 给出了 13 个设置坐标系标签的相关函数。

表 9.13 plt 库的标签设置函数（共 13 个）

函　　数	描　　述
plt.figlegend(handles, label, loc)	为全局绘图区域放置图注
plt.legend()	为当前坐标图放置图注
plt.xlabel(s)	设置当前 x 轴的标签
plt.ylabel(s)	设置当前 y 轴的标签
plt.xticks(array, 'a', 'b', 'c')	设置当前 x 轴刻度位置的标签和值
plt.yticks(array, 'a', 'b', 'c')	设置当前 y 轴刻度位置的标签和值
plt.clabel(cs,v)	为等值线图设置标签
plt.get_figlabels()	返回当前绘图区域的标签列表
plt.figtext(x, y, s, fontdic)	为全局绘图区域添加文字
plt.title()	设置标题
plt.suptitle()	为当前绘图区域添加中心标题
plt.text(x, y, s, fontdic, withdash)	为坐标图轴添加注释
plt.annotate(note, xy, xytext, xycoords, textcoords, arrowprops)	用箭头在指定数据点创建一个注释或一段文本

【微实例 9.2】带标签的坐标系。

绘制一个带标签的坐标系，代码如下，绘制效果如图 9.5 所示。

源代码 9-3：带标签的坐标系绘制

微实例 9.2　　m9.2PlotCoordinate.py

```
1   import matplotlib.pyplot as plt
2   import matplotlib
3   matplotlib.rcParams['font.family']='SimHei'
4   matplotlib.rcParams['font.sans-serif'] = ['SimHei']
5   plt.plot([1,2,4], [1,2,3])
6   plt.title("坐标系标题")
7   plt.xlabel('时间 (s)')
8   plt.ylabel('范围 (m)')
9   plt.xticks([1,2,3,4,5],[r'$\pi/3$', r'$2\pi/3$', r'$\pi$',\
                        r'$4\pi/3$', r'$5\pi/3$'])
10  plt.show()
```

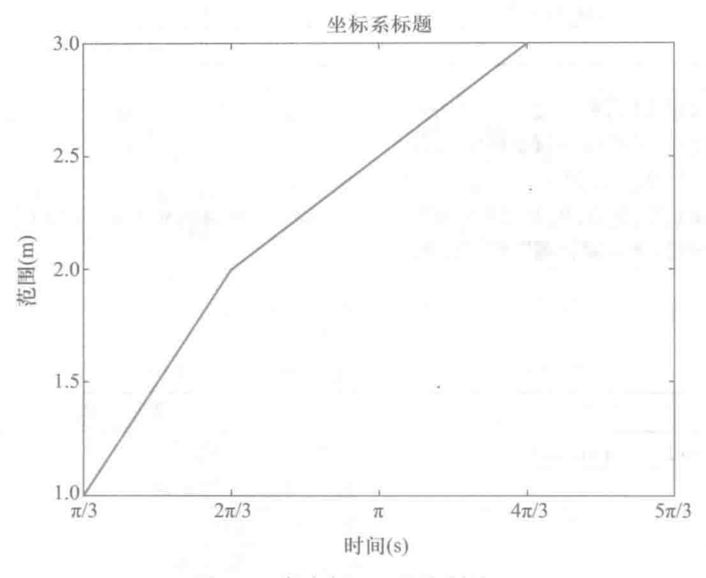


图 9.5　微实例 9.2 的绘制效果

plt 库提供了 3 个区域填充函数，对绘图区域填充颜色，如表 9.14 所示。

表 9.14　plt 库的区域填充函数（共 3 个）

函　　数	描　　述
fill(x,y,c,color)	填充多边形
fill_between(x,y1,y2,where,color)	填充两条曲线围成的多边形
fill_betweenx(y,x1,x2,where,hold)	填充两条水平线之间的区域

【微实例 9.3】带局部阴影的坐标系。

绘制一个带局部阴影的坐标系，代码如下，绘制效果如图 9.6 所示。

微实例 9.3　　m9.3PlotDarkCoordinate.py




```
1   import matplotlib.pyplot as plt
2   import numpy as np
3   x = np.linspace(0, 10, 1000)
4   y = np.cos(2*np.pi*x) * np.exp(-x)+0.8
5   plt.plot(x,y,'k',color='r',label="$exp-decay$",linewidth=3)
6   plt.axis([0,6,0,1.8])
7   ix = (x>0.8) & (x<3)
8   plt.fill_between(x, y ,0, where = ix,\
                           facecolor='grey', alpha=0.25)
9   plt.text(0.5*(0.8+3), 0.2, r"$\int_a^b f(x)\mathrm{d}x$",\
                    horizontalalignment='center')
10  plt.legend()
11  plt.show()
```

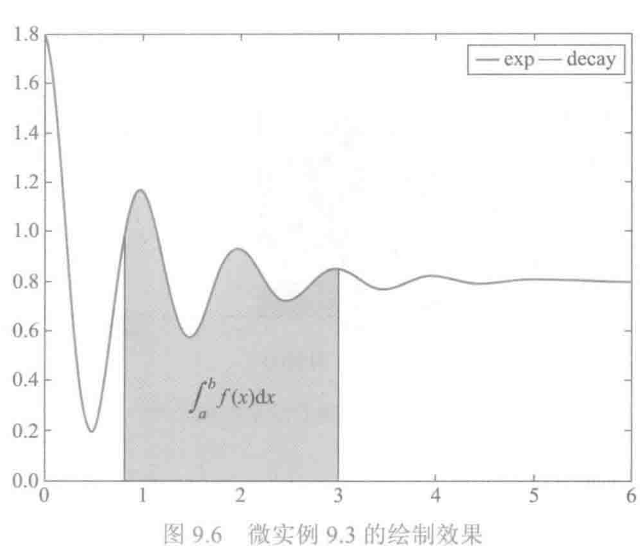


图 9.6　微实例 9.3 的绘制效果

思考与练习

9.9　如何在一个绘制区域中绘制上下两个坐标系？

9.10　一个专业的科学坐标系都有哪些组成部分？

9.11　利用 matplotlib 绘制函数曲线的最精简代码是什么？

9.5　实例 18：科学坐标图绘制

要点： 使用 matplotlib.pyplot 子库绘制科学坐标图并适当标注。

采用坐标系绘制和展示数据趋势是经常使用的功能，掌握绘制专业的科学坐标系将会为生活和工作提供更高效的支持。

科学坐标图有 4 个要素：坐标轴、数据曲线、标题和图注。这个实例以阻尼衰减曲线绘制来具体阐述科学坐标系的绘制方法。

本实例同时展示了在同一个区域用不同颜色和线条绘制两种曲线的方法，两条曲线分别为(x,y)和(x,z)，绘制效果如图 9.7 所示。

```
x = np.linspace(0.0, 6.0, 100)
y = np.cos(2 * np.pi * x) * np.exp(-x)+0.8
z = 0.5 * np.cos(x ** 2)+0.8
```

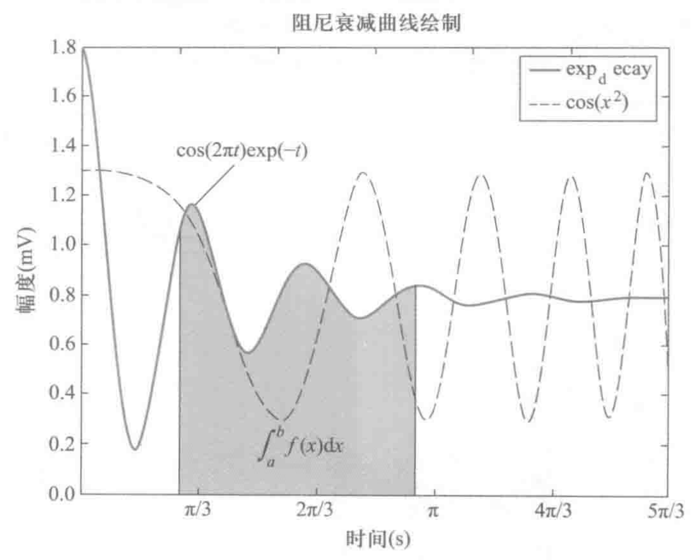


图 9.7 阻尼衰减曲线坐标图绘制

实例代码 18.1 如下：

源代码 9-5：阻尼衰减曲线坐标图绘制

实例代码 18.1　　e18.1PlotDamping.py

```
1   #e20.1PlotDamping.py
2   import numpy as np
3   import matplotlib.pyplot as plt
4   import matplotlib
5   matplotlib.rcParams['font.family']='SimHei'
6   matplotlib.rcParams['font.sans-serif'] = ['SimHei']
7   def Draw(pcolor, nt_point, nt_text, nt_size):
8       plt.plot(x, y, 'k', label="$exp_decay$", color=pcolor,\
                 linewidth=3, linestyle="-")
9       plt.plot(x, z, "b--", label="$cos(x^2)$", linewidth=1)
10      plt.xlabel('时间(s)')
11      plt.ylabel('幅度(mV)')
12      plt.title("阻尼衰减曲线绘制")
```

```
13      plt.annotate('$\cos(2 \pi t) \exp(-t)$', xy=nt_point, \
            xytext=nt_text,fontsize=nt_size,arrowprops= \
            dict(arrowstyle='->', connectionstyle="arc3,rad=.1"))
14  def Shadow(a, b):
15      ix = (x>a) & (x<b)
16  plt.fill_between(x,y,0,where=ix,facecolor='grey',alpha=0.25)
17      plt.text(0.5 * (a + b), 0.2, r"$\int_a^b f(x)\mathrm{d}x$",\
                 horizontalalignment='center')
18  def XY_Axis(x_start, x_end, y_start, y_end):
19      plt.xlim(x_start, x_end)
20      plt.ylim(y_start, y_end)
21      plt.xticks([np.pi/3, 2 * np.pi/3, 1 * np.pi, 4 * np.pi/3,\
        5*np.pi/3],['$\pi/3$','$2\pi/3$','$\pi$','$4\pi/3$','$5\pi/3$'])
22  x = np.linspace(0.0, 6.0, 100)
23  y = np.cos(2 * np.pi * x) * np.exp(-x)+0.8
24  z = 0.5 * np.cos(x ** 2)+0.8
25  note_point,note_text,note_size=(1,np.cos(2*np.pi)*\
26  np.exp(-1)+0.8),(1, 1.4), 14
27  fig = plt.figure(figsize=(8, 6), facecolor="white")
28  plt.subplot(111)
29  Draw("red", note_point, note_text, note_size)
30  XY_Axis(0, 5, 0, 1.8)
31  Shadow(0.8, 3)
32  plt.legend()
33  plt.savefig('sample.JPG')
34  plt.show()
```

为了便于阅读，程序设计了 Draw()、Shadow()和 XY_Axis()函数，分别用于绘制曲线、设置阴影和修改坐标轴。第 13 行 annotate()函数配合箭头在曲线绘图界面添加动态注释，箭头的线条和尖端都有多种样式和参数，可以通过 arrowprops 和 connectionstyle 选择，具体请参考官方文档。plt.savefig()函数能够将产生的坐标图保存为文件。

实例代码 18.1 中所有代码均已在 9.4 节中介绍，请读者阅读并运行代码，理解编写科学坐标系的方法。

拓展： 科学计算可视化

可视化技术与科学计算相结合形成了可视化技术的一个重要分支——科学计算可视化(Visualization in Scientific Computing)。科学计算可视化将科学数据如测量获得的数值、图像或计算产生的数字信息等以直观的图形图像方式展示。通过直观展示，宇宙空间有了颜色，物理现象更为直观，感性理解和理性求证相辅相成，共同促进科学计算的深入发展。

思考与练习

9.12 解释实例代码 18.1 第 13 行代码的含义。

9.13 解释实例代码 18.1 第 32 行图注中的内容在哪里设定。

9.14 解释实例代码 18.1 绘制曲线的颜色和线形在哪里设定。

9.6 实例 19：多级雷达图绘制

要点： 使用 matplotlib.pyplot 绘制圆形多级雷达图，在展示对象多属性的差异。

雷达图是通过多个离散属性比较对象的最直观工具，掌握绘制雷达图的方法将会为生活和工作带来乐趣。

游戏角色中经常出现表示人物能力值的雷达图。DOTAMAX 测试版曾经推出过显示玩家能力值分布的雷达图，只要点击自己或好友头像，就可以看到能力值在综合、KDA、发育、推进、生存、输出等方面的分布，如图 9.8 所示。

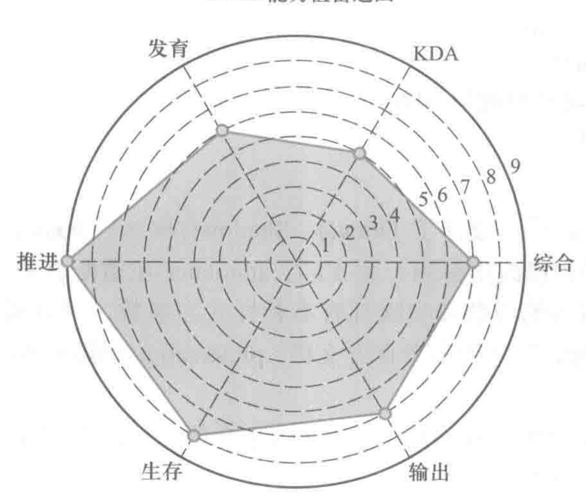


图 9.8 DOTA 人物能力值雷达图

源代码 9-6：DOTA 人物能力值雷达图绘制



本实例使用 Python 来绘制这样的多级雷达图，即在一组同心圆上填充不规则六边形，其每个顶点到圆心的距离代表人物数据的某个属性。实例代码 19.1 如下：

实例代码 19.1 e19.1DrawDotaRadar.py

```
1  #e21.1DrawDotaRadar.py
2  import numpy as np
3  import matplotlib.pyplot as plt
```

```
4   import matplotlib
5   matplotlib.rcParams['font.family']='SimHei'
6   matplotlib.rcParams['font.sans-serif'] = ['SimHei']
7   labels = np.array(['综合','KDA','发育','推进','生存','输出'])
8   nAttr = 6
9   data = np.array([7, 5, 6, 9, 8, 7]) #数据值
10  angles = np.linspace(0, 2*np.pi, nAttr, endpoint=False)
11  data = np.concatenate((data, [data[0]]))
12  angles = np.concatenate((angles, [angles[0]]))
13  fig = plt.figure(facecolor="white")
14  plt.subplot(111, polar=True)
15  plt.plot(angles,data,'bo-',color ='g',linewidth=2)
16  plt.fill(angles,data,facecolor='g',alpha=0.25)
17  plt.thetagrids(angles*180/np.pi, labels)
18  plt.figtext(0.52, 0.95, 'DOTA能力值雷达图', ha='center')
19  plt.grid(True)
20  plt.savefig('dota_radar.JPG')
21  plt.show()
```

实例代码 19.1 中第 4 到第 6 行用于支持中文。由于 DOTA 实例包含 6 个属性，设置属性标签 labels，并预设一组玩家数据 data。

np.linspaces()函数设定起点为 0、末值为 2π、返回一个两端点间数值平均分布的长为 nAttr 的数组 angles，它表示从一个属性点到下一个属性点笔画需要旋转的角度，它取决于属性 nAttr 的大小，也是雷达图的多边形边数。

np.concatenate()函数用于将数据和角度的数组首尾闭合起来，便于调用 plot()函数绘制。

建立基本绘图对象后，使用 subplot()函数建立极坐标系的子分区。polar 参数指定了绘制类型为极坐标，这是 subplot()除默认正方形坐标系外唯一支持的内置坐标图。建立极坐标后，使用 plot()函数依照 data 提供的数据画出不规则六边形，然后使用 fill()函数填充半透明颜色。thetagrids()函数为极坐标设置标签，这里把标签安放在六角形的顶点上，需要将角度数据和文字一起作为参数传给 thetagrids 函数。

除了 DOTA 游戏，雷达图应用广泛。再看一个例子。美国约翰霍普金斯大学霍兰德教授认为兴趣是人们活动的巨大动力，凡是具有职业兴趣的职业，都可以提高人们的积极性，促使人们积极地、愉快地从事该职业。因此，他研究了人格类型、兴趣与职业间的关系，提出了“霍兰德职业兴趣理论”，认为人格可分为现实型、研究型、艺术型、社会型、企业型和常规型 6 种类型。

拓展： 兴趣是最好的老师

爱因斯坦说过，“兴趣是最好的老师”。兴趣来源于好奇，好奇心则是人类与生俱来的。如果读者对本书设计的实例感到好奇，这说明，你已经找到了最好的老师。全书共 25 个实例，绝大部分为作者结合实际教学效果原创制作，它们将为读者带来学习 Python 语言的最佳体验。

展示霍兰德人格分析最有效的工具是雷达图。以工程师、实验员、艺术家、推销员、社会工作者、记事员 6 个职业数据为例，实例代码 19.2 给出了霍兰德人格分析绘制的雷达图，不同职业间的区别非常直观。实例代码 19.2 如下，运行结果如图 9.9 所示。

源代码 9-7：霍兰德人格分析雷达图绘制



实例代码 19.2 e19.2DrawHollandRadar.py

```
1  #e22.1DrawHollandRadar
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import matplotlib
5  matplotlib.rcParams['font.family']='SimHei'
6  matplotlib.rcParams['font.sans-serif'] = ['SimHei']
7  radar_labels = np.array(['研究型(I)','艺术型(A)','社会型(S)',\
                           '企业型(E)','常规型(C)','现实型(R)'])
8  nAttr = 6
9  data = np.array([[0.40, 0.32, 0.35, 0.30, 0.30, 0.88],
10                  [0.85, 0.35, 0.30, 0.40, 0.40, 0.30],
11                  [0.43, 0.89, 0.30, 0.28, 0.22, 0.30],
12                  [0.30, 0.25, 0.48, 0.85, 0.45, 0.40],
13                  [0.20, 0.38, 0.87, 0.45, 0.32, 0.28],
14                  [0.34, 0.31, 0.38, 0.40, 0.92, 0.28]]) #数据值
15 data_labels=('工程师','实验员','艺术家','推销员','社会工作者','记事员')
16 angles = np.linspace(0, 2*np.pi, nAttr, endpoint=False)
17 data = np.concatenate((data, [data[0]]))
18 angles = np.concatenate((angles, [angles[0]]))
19 fig = plt.figure(facecolor="white")
20 plt.subplot(111, polar=True)
21 plt.plot(angles,data,'bo-',color ='gray',linewidth=1,alpha=0.2)
22 
23 plt.plot(angles,data,'o-', linewidth=1.5, alpha=0.2)
24 plt.fill(angles,data, alpha=0.25)
25 plt.thetagrids(angles*180/np.pi, radar_labels,frac = 1.2)
26 plt.figtext(0.52, 0.95, '霍兰德人格分析', ha='center', size=20)
27 legend=plt.legend(data_labels,loc=(0.94,0.80),labelspacing=0.1)
28 plt.setp(legend.get_texts(), fontsize='small')
29 plt.grid(True)
30 plt.savefig('holland_radar.JPG')
31 plt.show()
```

实例代码 19.1 和实例代码 19.2 分别给出了绘制单一数据雷达图和多数据雷达图的方法，是不是很有趣？

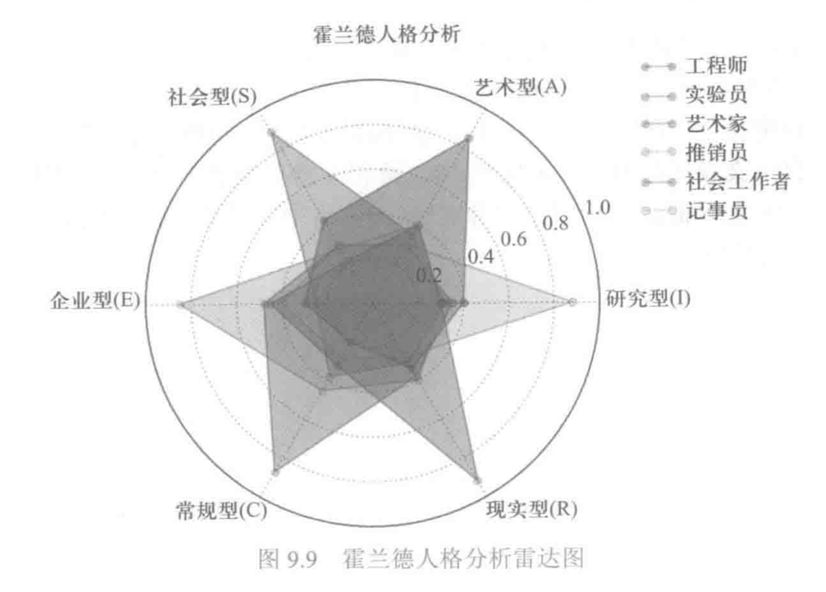


图 9.9　霍兰德人格分析雷达图

思考与练习

9.15　解释实例代码 19.1 第 14 行代码的含义。

9.16　解释实例代码 19.1 第 18 行代码的含义。

9.17　解释实例代码 19.2 第 27 行代码的含义。

本 章 小 结

本章以科学计算和可视化为中心，介绍了两个强大的工具库 numpy 和 matplotlib.pyplot，通过生成手绘风格图片、绘制科学坐标系和绘制雷达图等实例展示了 Python 在科学计算方面的强大功能。

程序练习题



9.1　方波绘制。在信号处理理论中，方波可近似表示为多个正弦波的叠加。事实上，任意一个方波信号都可以使用傅里叶变换为多个正弦波表示。利用 numpy 和 matplotlib 在坐标系中绘制方波的无穷级数表示。请尝试调节正弦波的个数、幅度以及周期，尽可能使方波边缘平滑。方波无穷级数表达式如下：

$$f = \sum_{k=1}^{\infty} \frac{4\sin(2k-1)t}{(2k-1)\pi}$$

9.2　心脏线绘制。笛卡儿心形线也称为心脏线，它是有一个尖点的外摆线。当

一个圆沿着另一个半径相同的圆滚动时，圆上一点的轨迹就是心脏线。请调研笛卡儿心形线，并使用 numpy 和 matplotlib 绘制一条笛卡儿心形线。

9.3 自定义手绘风。修改实例 17，使手绘效果更符合你的审美特点。

9.4 自定义规律绘制。参考实例 18，绘制你感兴趣的一个数学或物理规律。

9.5 乒乓选手雷达图绘制。参考实例 19，为中国乒乓球选手绘制雷达图，至少建立 4 个属性值。

第 10 章　网络爬虫和自动化

电子教案 10-1：网络爬虫和自动化



计算机会按照你给出的指令去执行，这是好事，但也是坏事。

The good news about computers is that they do what you tell them to do. The bad news is that they do what you tell them to do.

——泰德·尼尔森（Ted Nelson）

信息技术先驱、超文本传输协议（HTTP）的设计者、哲学家、社会学家

学习目标

（1）掌握网络爬虫的基本方法。
（2）运用 requests 库编写基本 URL 访问过程。
（3）运用 beautifulsoup4 库解析和处理 HTML。
（4）掌握向搜索引擎自动提交关键词并获取返回结果的方法。

似乎一瞬间全世界都开始讨论网络爬虫，随着网络的迅速发展，如何有效地提取并利用信息很大程度上决定了解决问题的效率。搜索引擎作为辅助程序员检索信息的工具已经有些力不从心。为了更高效地获取指定信息需要定向抓取并分析网页资源，网络爬虫火爆了起来。

本章将讲述编写优质网络爬虫的方法，嗯，别人有的这里也有，而且还更好。

10.1 问题概述

要点： Python 语言实现网络爬虫的问题引入。

Python 语言发展中有一个里程碑式的应用事件，即美国谷歌（Google）公司在搜索引擎后端采用 Python 语言进行链接处理和开发，这是该语言发展成熟的重要标志。Python 语言的简洁性和脚本特点非常适合链接和网页处理，因此，在 Python 的计算生态中，与 URL 和网页处理相关的第三方库很多。

万维网（WWW）的快速发展带来了大量获取和提交网络信息的需求，这产生了“网络爬虫”等一系列应用。Python 语言提供了很多类似的函数库，包括 urllib、urllib2、urllib3、wget、scrapy、requests 等。这些库作用不同、使用方式不同、用户体验不同。对于爬取回来的网页内容，可以通过 re（正则表达式）、beautifulsoup4 等函数库来处理。随着该领域各函数库的发展，本章将详细介绍其中最重要且最主流的两个函数库：requests 和 beautifulsoup4，它们都是第三方库。

除了网络爬虫，自动向网站提交数据既有趣又有用，这样的功能也能通过 requests 库实现。本章将以两个类似例子介绍从网络获取数据及向网络提交数据的方法。

网络爬虫应用一般分为两个步骤：①通过网络链接获取网页内容；②对获得的网页内容进行处理。这两个步骤分别使用不同的函数库：requests 和 beautifulsoup4。

采用 pip 指令安装 requests 库，如果在 Python 2 和 Python 3 并存的系统中，采用 pip3 指令，代码如下：

```
:\>pip install requests   # 或者 pip3 install requests
```

采用 pip 或 pip3 指令安装 beautifulsoup4 库，注意，不要安装 beautifulsoup 库，后者由于年久失修，已经不再维护了。安装命令如下：

```
:\>pip install beautifulsoup4   # 或者 pip3 install beautifulsoup4
```

使用 Python 语言实现网络爬虫和信息提交是非常简单的事情，代码行数很少，也无须掌握网络通信等方面的知识，非常适合非专业读者使用。然而，肆意地爬取网络数据并不是文明现象，通过程序自动提交内容争取竞争性资源也不公平。就像那些肆意的推销电话一样，他们无视接听者意愿，不仅令人讨厌也有可能引发法律纠纷。

拓展： Robots 排除协议

Robots 排除协议（Robots Exclusion Protocol），也被称为爬虫协议，它是网站管理者表达是否希望爬虫自动获取网络信息意愿的方法。管理者可以在网站根目

录放置一个 robots.txt 文件，并在文件中列出哪些链接不允许爬虫爬取。一般搜索引擎的爬虫会首先捕获这个文件，并根据文件要求爬取网站内容。Robots 排除协议重点约定不希望爬虫获取的内容，如果没有该文件则表示网站内容可以被爬虫获得，然而，Robots 协议不是命令和强制手段，只是国际互联网的一种通用道德规范。绝大部分成熟的搜索引擎爬虫都会遵循这个协议，建议个人也能按照互联网规范要求合理使用爬虫技术。

思考与练习

10.1　请思考网络爬虫的可能应用。

10.2　模块 10：requests 库的使用

要点：requests 库是一个简洁且简单的处理 HTTP 请求的第三方库。

10.2.1　requests 库概述

requests 库是一个简洁且简单的处理 HTTP 请求的第三方库，它的最大优点是程序编写过程更接近正常 URL 访问过程。这个库建立在 Python 语言的 urllib3 库的基础上，类似这种在其他函数库之上再封装功能、提供更友好函数的方式在 Python 语言中十分常见。在 Python 生态圈里，任何人都有通过技术创新或体验创新发表意见和展示才华的机会。

图片资料 10-1：Python 快速参考之 requests、bs4 库



requests 库支持非常丰富的链接访问功能，包括国际域名和 URL 获取、HTTP 长连接和连接缓存、HTTP 会话和 Cookie 保持、浏览器使用风格的 SSL 验证、基本的摘要认证、有效的键值对 Cookie 记录、自动解压缩、自动内容解码、文件分块上传、HTTP(S)代理功能、连接超时处理、流数据下载等。有关 requests 库的更多介绍请访问 http://docs.python-requests.org。

10.2.2　requests 库解析

网络爬虫和信息提交只是 requests 库能支持的基本功能，本节重点介绍与这两个功能相关的一些常用函数。其中，与网页请求相关的函数如表 10.1 所示。

表 10.1　requests 库中的网页请求函数（共 6 个）

函　　数	描　　述
get(url [, timeout=n])	对应于 HTTP 的 GET 方式，获取网页最常用的方法，可以增加 timeout=n 参数，设定每次请求超时时间为 n 秒

续表

函 数	描 述
post(url, data = {'key':'value'})	对应于 HTTP 的 POST 方式，其中字典用于传递客户数据
delete(url)	对应于 HTTP 的 DELETE 方式
head(url)	对应于 HTTP 的 HEAD 方式
options(url)	对应于 HTTP 的 OPTIONS 方式
put(url, data = {'key':'value'})	对应于 HTTP 的 PUT 方式，其中字典用于传递客户数据

get()是获取网页最常用的方式，在调用 requests.get()函数后，返回的网页内容会保存为一个 Response 对象，其中，get()函数的参数 url 链接必须采用 HTTP 或 HTTPS 方式访问，例如：

```
>>>import requests
>>> r = requests.get("http://www.baidu.com") #使用 get 方法打开百度链接
>>>type(r)
<class 'requests.models.Response'>              #返回 Response 对象
```

读者在深入了解 HTTP 协议后会很快理解上述函数的用法和意义，本书不过多介绍，从爬虫应用角度来看，只需要掌握 get()函数即可获取网页。

和浏览器的交互过程一样，requests.get()代表请求过程，它返回的 Response 对象代表响应。返回内容作为一个对象更便于操作，Response 对象的属性如表 10.2 所示，需要采用<a>.<b>形式。

表 10.2 Response 对象的属性（共 4 个）

属 性	描 述
status_code	HTTP 请求的返回状态，整数，200 表示连接成功，404 表示失败
text	HTTP 响应内容的字符串形式，即 url 对应的页面内容
encoding	HTTP 响应内容的编码方式
content	HTTP 响应内容的二进制形式

status_code 属性返回请求 HTTP 后的状态，在处理数据之前要先判断状态情况，如果请求未被响应，需要终止内容处理。text 属性是请求的页面内容，以字符串形式展示。encoding 属性非常重要，它给出了返回页面内容的编码方式，可以通过对 encoding 属性赋值更改编码方式，以便于处理中文字符。content 属性是页面内容的二进制形式。例如：

```
>>>r = requests.get("http://www.baidu.com")
>>>r.status_code        #返回状态
200
>>>r.text               #观察返回的内容，中文字符是否能正常显示
(输出略)
>>>r.encoding           #默认的编码方式是 ISO-8859-1，所以中文是乱码
'ISO-8859-1'
>>> r.encoding = 'utf-8'      # 更改编码方式为 utf-8
```

```
>>> r.text              # 更改完成，返回内容中的中文字符可以正常显示了
(输出略)
```

除了属性，Response 对象还提供一些方法，如表 10.3 所示。

表 10.3　Response 对象的方法（共 2 个）

方　法	描　述
json()	如果 HTTP 响应内容包含 JSON 格式数据，则该方法解析 JSON 数据
raise_for_status()	如果不是 200，则产生异常

json()方法能够在 HTTP 响应内容中解析存在的 JSON 数据，这将带来解析 HTTP 的便利。raise_for_status()方法能在非成功响应后产生异常，即只要返回的请求状态 status_code 不是 200，这个方法会产生一个异常，用于 try-except 语句。使用异常处理语句可以避免设置一堆复杂的 if 语句，只需要在收到响应时调用这个方法，就可以避开状态字 200 以外的各种意外情况。

requests 会产生几种常用异常。当遇到网络问题时，如 DNS 查询失败、拒绝连接等，requests 会抛出 ConnectionError 异常；遇到无效 HTTP 响应时，requests 则会抛出 HTTPError 异常；若请求 url 超时，则抛出 Timeout 异常；若请求超过了设定的最大重定向次数，则会抛出一个 TooManyRedirects 异常。

获取一个网页内容的函数建议采用如下代码的第 2 到第 9 行，第 10 和第 11 行是测试代码。

源代码 10-1：获得一个 HTML 页面的通用代码



```
1   import requests
2   def getHTMLText():
3       try:
4           r = requests.get(url, timeout=30)
5           r.raise_for_status() #如果状态不是 200，引发异常
6           r.encoding = 'utf-8' #无论原来用什么编码，都改成 utf-8
7           return r.text
8       except:
9           return ""
10  url = "http://www.baidu.com"
11  print(getHTMLText(url))
```

拓展： HTTP 的 GET 和 POST

HTTP 协议定义了客户端与服务器交互的不同方法，最基本的方法是 GET 和 POST。顾名思义，GET 可以根据某链接获得内容，POST 用于发送内容。然而，GET 也可以向链接提交内容，与 POST 的区别如下。

（1）GET 方式可以通过 URL 提交数据，待提交数据是 URL 的一部分；采用 POST 方式，待提交数据放置在 HTML HEADER 内。

（2）GET 方式提交的数据最多不超过 1024 字节，POST 没有对提交内容的长度限制。

（3）安全性问题。使用 GET 时参数会显示在 URL 中，而 POST 不会。所以，如果这些数据是非敏感数据，那么使用 GET；如果提交数据是敏感数据，建议采用 POST 方式。

思考与练习

10.2 请查阅资料更多了解 HTTP 协议中 post 和 get 功能的区别和联系。

10.3 requests 库提供了一个 post()函数，请查阅资料了解该函数的用法。

10.4 请用 get()访问百度页面，用 len()函数分别计算 text 属性和 content 属性所返回网页内容的长度，思考产生长度差异的原因。

10.3 模块 11：beautifulsoup4 库的使用

要点：beautifulsoup4 库是一个解析和处理 HTML 和 XML 的第三方库。

10.3.1 beautifulsoup4 库概述

图片资料 10-1：Python 快速参考之 requests、bs4 库



使用 requests 库获取 HTML 页面并将其转换成字符串后，需要进一步解析 HTML 页面格式，提取有用信息，这需要处理 HTML 和 XML 的函数库。

beautifulsoup4 库，也称为 Beautiful Soup 库或 bs4 库，用于解析和处理 HTML 和 XML。需要注意的是，它不是 BeautifulSoup 库。它的最大优点是能根据 HTML 和 XML 语法建立解析树，进而高效解析其中的内容。

HTML 建立的 Web 页面一般非常复杂，除了有用的内容信息外，还包括大量用于页面格式的元素，直接解析一个 Web 网页需要深入了解 HTML 语法，而且比较复杂。beautifulsoup4 库将专业的 Web 页面格式解析部分封装成函数，提供了若干有用且便捷的处理函数。

beautifulsoup4 库采用面向对象思想实现，简单地说，它把每个页面当作一个对象，通过<a>.<b>的方式调用对象的属性（即包含的内容），或者通过<a>.<b>()的方式调用方法（即处理函数）。

在使用 beautifulsoup4 库之前，需要进行引用，由于这个库的名字非常特殊且采用面向对象方式组织，可以用 from-import 方式从库中直接引用 BeautifulSoup 类，方法如下：

```
>>>from bs4 import BeautifulSoup
```

有关 beautifulsoup4 库的更多介绍请参考这个第三方库主页：http://www.crummy.com/software/BeautifulSoup/bs4/。这里主要介绍一些常用方法，辅助读者更好理解本

章后续实例。

10.3.2　beautifulsoup4 库解析

beautifulsoup4 库中最主要的是 BeautifulSoup 类，每个实例化的对象相当于一个页面。采用 from-import 导入库中的 Beautifulsoup 类后，使用 BeautifulSoup()创建一个 BeautifulSoup 对象。

```
>>>import requests
>>>from bs4 import BeautifulSoup
>>>r = requests.get("http://www.baidu.com")
>>>r.encoding = "utf-8"          #为了简化代码，没有考虑异常情况
>>>soup = BeautifulSoup(r.text)   #soup 就是一个 BeautifulSoup 对象
>>>type(soup)
<class 'bs4.BeautifulSoup'>
```

创建的 BeautifulSoup 对象是一个树形结构，它包含 HTML 页面中的每一个 Tag（标签）元素，如<head>、<body>等。具体来说，HTML 中的主要结构都变成了 BeautifulSoup 对象的一个属性，可以直接用<a>.<b>形式获得，其中<b>的名字采用 HTML 中标签的名字。表 10.4 列出了 BeautifulSoup 中常用的一些属性，例子如下。

表 10.4　BeautifulSoup 对象的常用属性（共 6 个）

属　性	描　述
head	HTML 页面的<head>内容
title	HTML 页面标题，在<head>之中，由<title>标记
body	HTML 页面的<body>内容
p	HTML 页面中第一个<p>内容
strings	HTML 页面所有呈现在 Web 上的字符串，即标签的内容
stripped_strings	HTML 页面所有呈现在 Web 上的非空格字符串

```
>>>soup.head  #略去<style>标签输出
<head><meta content="text/html;charset=utf-8" http-equiv="content-type"><meta content="IE=Edge" http-equiv="X-UA-Compatible"><title>百度一下，你就知道</title></meta></meta></head>
>>>title=soup.title
<title>百度一下，你就知道</title>
>>>type(title)   #每个对应 HTML Tag 的属性是一个 Tag 类型
<class 'bs4.element.Tag'>
>>>soup.p
<p id="lh"><a href="http://www.baidu.com/cache/sethelp/help.html" target="_blank">把百度设为主页</a><a href="http://home.baidu.com">关于百度</a><a href="http://ir.baidu.com">About Baidu</a></p>
```

BeautifulSoup 属性与 HTML 的标签名称相同，远不止表 10.4 中的这些，更多内容请读者结合 HTML 语法理解。每一个 Tag 标签在 beautifulsoup4 库中也是一个

对象，称为 Tag 对象。上例中，title 是一个标签对象。每个标签对象在 HTML 中都有类似的结构：

```
<a class="mnav" href="http://www.nuomi.com">糯米</a>
```

其中，尖括号（<>）中标签的名字是 name，尖括号内其他项是 attrs，尖括号之间的内容是 string。因此，可以通过 Tag 对象的 name、attrs 和 string 属性获得相应内容，采用<a>.<b>的语法形式。标签 Tag 有 4 个常用属性，如表 10.5 所示，例子如下。

表 10.5 标签对象的常用属性（共 4 个）

属　性	描　述
name	字符串，标签的名字，比如 div
attrs	字典，包含了原来页面 Tag 所有的属性，比如 href
contents	列表，这个 Tag 下所有子 Tag 的内容
string	字符串，Tag 所包围的文本，网页中真实的文字

```
>>>soup.a
<a class="mnav" href="http://www.nuomi.com">糯米</a>
>>>soup.a.name
'a'
>>>soup.a.attrs
{'href': 'http://www.nuomi.com', 'class': ['mnav']}
>>>soup.a.string
'糯米'
>>>title.name  #title 变量在上段例子中已经定义
'title'
>>>title.string
'百度一下，你就知道'
>>>soup.p.contents
[<a    href="http://www.baidu.com/cache/sethelp/help.html"    target=
"_blank">把百度设为主页</a>, <a href="http://home.baidu.com">关于百度
</a>, <a href="http://ir.baidu.com">About Baidu</a>]
```

由于 HTML 语法可以在标签中嵌套其他标签，所以，string 属性的返回值遵循如下原则。

（1）如果标签内部没有其他标签，string 属性返回其中的内容。

（2）如果标签内部还有其他标签，但只有一个标签，string 属性返回最里面标签的内容。

（3）如果标签内部有超过 1 层嵌套的标签，string 属性返回 None（空字符串）。

HTML 语法中同一个标签会有很多内容，例如<a>标签，百度首页一共有 13 处，列表如下，直接调用 soup.a 只能返回第一个。

```
<a class="mnav" href="http://www.nuomi.com">糯米</a>
<a class="mnav" href="http://news.baidu.com">新闻</a>
<a class="mnav" href="http://www.hao123.com">hao123</a>
```

```
<a class="mnav" href="http://map.baidu.com">地图</a>
<a class="mnav" href="http://v.baidu.com">视频</a> …
```

当需要列出标签对应的所有内容或者需要找到非第一个标签时，需要用到 BeautifulSoup 的 find()和 find_all()方法。这两个方法会遍历整个 HTML 文档，按照条件返回标签内容。

BeautifulSoup.find_all(name, attrs, recursive, string, limit)

作用：根据参数找到对应标签，返回列表类型。

参数如下。

name：按照 Tag 标签名字检索，名字用字符串形式表示，例如 div、li。

attrs：按照 Tag 标签属性值检索，需要列出属性名称和值，采用 JSON 表示。

recursive：设置查找层次，只查找当前标签下一层时使用 recursive=False。

string：按照关键字检索 string 属性内容，采用 string=开始。

limit：返回结果的个数，默认返回全部结果。

```
>>>a = soup.find_all('a')       #查找所有的<a>
>>> len(a)
13
>>>soup.find_all('script')
[<script>var md5="230CFBXBZBXCCCDBYCEDREADTEHDREIDZ"</script>, <script
src="http://www.zgxiangxin.com/jquery/jquery-1.9.4.min.
js"></script>]
>>>soup.find_all('script',{'src':'http://www.zgxiangxin.com/\
jquery/jquery-1.9.4.min.js'})  #筛选，只查找 src=字符和串的标签
[<script src="http://www.zgxiangxin.com/jquery/jquery-1.9.4
.min.js"></script>]
>>>import re   # 使用正则表达式库，可以用这个库实现字符串片段匹配
>>>soup.find_all('script',{'src':re.compile('jquery')})
[<script src="http://www.zgxiangxin.com/jquery/jquery-1.9.4
.min.js"></script>]
>>>soup.find_all(string=re.compile('百度'))
['百度一下，你就知道', '把百度设为主页', '关于百度', '使用百度前必读']
```

简单地说，BeautifulSoup 的 find_all()方法可以根据标签名字、标签属性和内容检索并返回标签列表，通过片段字符串检索时需要使用正则表达式 re 函数库，re 是 Python 标准库，直接通过 import re 即可使用。采用 re.compile('**jquery**')实现对片段字符串（如'**jquery**'）的检索。当对标签属性检索时，属性和对应的值采用 JSON 格式，例如：

'src':re.compile('jquery')

其中，键值对中值的部分可以是字符串或者正则表达式。

拓展： 正则表达式

正则表达式是字符串的一种逻辑表达，一般在计算机编译器中使用。Python 语言采用正则表达式辅助字符串查找。正则表达式是一种规则，只要字符串符合这个规则，就算作匹配。例如，通过 re.compile()函数注册一个正则表达式'jquery',

则所有包含该表达式的字符串都与它匹配。除了字符串，正则表达式还可以通过*+{}等符号扩展功能。有兴趣的读者可以查阅资料了解 Python 中正则表达式函数库 re 的更多高级使用。

除了 find_all()方法，BeautifulSoup 类还提供一个 find()方法，它们的区别只是前者返回全部结果而后者返回找到的第一个结果，find_all()函数由于可能返回更多结果，所以采用列表形式；find()函数返回字符串形式。

BeautifulSoup.find(name, attrs, recursive, string)

作用：根据参数找到对应标签，采用字符串返回找到的第一个值。

参数：与 find_all()方法一样，略。

处理网页需要对 HTML 有一定的理解，然而实现爬虫并不算复杂，这里仅介绍 beautifulsoup4 库中与爬虫相关的一些属性和操作。beautifulsoup4 库是一个非常完备且活跃的 HTML 解析函数库，它还可以完成更多复杂操作。深入使用请阅读 beautifulsoup4 库官方网站上提供的使用文档。

思考与练习

10.5　下列不属于 HTML 的 Tag 的是（　　）。

A．title　　B．a　　C．class　　D．head

10.6　这是一个简单的 HTML 页面，请保存为字符串，完成后面的计算要求。

```
<html>
    <head>
    <title>Simple test</title>
    </head>
    <body>
    <p id="China">中国，<b>你好！</b>。 </p>
    <p id="World">世界，<b>大同！</b>。 </p>
    </body>
</html>
```

（1）打印 head 标签的内容。

（2）获取 body 标签的内容。

（3）获取 id 为 China 的标签对象。

（4）获取并打印 HTML 页面中的中文字符。

10.4　实例 20：中国大学排名爬虫

要点： 这是一个获取中国大学排名的爬虫实例，采用了 requests 和 beautifulsoup4 函数库。

“有人的地方就有江湖”，有大学的地方就有排名。尽管中国大学排名不能客观反映各高校的绝对实力和影响力，但也能反映大学间的相对关系。

全球有很多份大学排名，这里以上海交通大学研发的“软科中国最好大学排名2016”为例，编写“大学排名爬虫”，如图 10.1 所示，从网络上获取数据，这个大学排名网址为 http://www.zuihaodaxue.cn/zuihaodaxuepaiming2016.html。拟从该网址爬取该名单上 310 所国内大学的排名数据，并将它们打印出来。读者可以对这些数据开展其他操作。

排名	学校名称	省市	总分	指标得分 生源质量（新生高考成绩得分）
1	清华大学	北京市	95.9	100.0
2	北京大学	北京市	82.6	98.9
3	浙江大学	浙江省	80	88.8
4	上海交通大学	上海市	78.7	90.6
5	复旦大学	上海市	70.9	90.4
6	南京大学	江苏省	66.1	90.7
7	中国科学技术大学	安徽省	65.5	90.1
8	哈尔滨工业大学	黑龙江省	63.5	80.9
9	华中科技大学	湖北省	62.9	83.5
10	中山大学	广东省	62.1	81.8

图 10.1　软科中国大学排名 2016 网页片段

大学排名爬虫的构建需要 3 个步骤：第一，从网络上获取网页内容；第二，分析网页内容并提取有用数据到恰当的数据结构中；第三，利用数据结构展示或进一步处理数据。由于大学排名是一个典型的二维数据，因此，结合 7.4 节介绍，采用二维列表存储该排名所涉及的表单数据。

具体来说，采用 requests 库爬取网页内容，使用 beautifulsoup4 库分析网页中的数据，提取 310 个学校的排名及相关数据，存储到二维列表中，最后采用用户偏好的方式打印出来。

为了解析网页上的数据，首先需要程序编写者观察爬虫页面的特点，即找到拟获取数据在 HTML 页面中的格式。打开大学排名页面，在浏览器菜单中选择“查看网页源代码”命令，该选项在所有浏览器中都存在，得到的 HTML 源代码如图 10.2 所示（为了便于阅读，该源代码做过一定排版）。

对比图 10.1 和图 10.2，每个大学排名的数据信息被封装在一个<tr></tr>之间的结构中。这是 HTML 语言表示表格中一行的标签，在这行中，每列内容采用<td></td>表示。以“清华大学”为例，它对应一行信息的 HTML 代码如下：

```
<tbody class="hidden_zhpm" style="text-align:center;">
  <tr class="alt">
      <td>1</td><td><div align="left">清华大学</div></td> <td>北京市</td><td>95.9</td><td class="hidden-xs need-hidden
      indicator5">100.0</td><td class="hidden-xs need-hidden indicator6"  style="display:none;">97.90%</td><td class=
      "hidden-xs need-hidden indicator7"  style="display:none;">37342</td><td class="hidden-xs need-hidden indicator8"
      style="display:none;">1.298</td><td class="hidden-xs need-hidden indicator9"  style="display:none;">1177</td><td
      class="hidden-xs need-hidden indicator10"  style="display:none;">109</td><td class="hidden-xs need-hidden
      indicator11"  style="display:none;">1137711</td><td class="hidden-xs need-hidden indicator12"  style="display:none;"
      >1187</td><td class="hidden-xs need-hidden indicator13"  style="display:none;">593522</td>
  </tr>
  <tr>
      <td>2</td><td><div align="left">北京大学</div></td> <td>北京市</td><td>82.6</td><td class="hidden-xs need-hidden
      indicator5">98.9</td><td class="hidden-xs need-hidden indicator6"  style="display:none;">95.96%</td><td class=
      "hidden-xs need-hidden indicator7"  style="display:none;">36137</td><td class="hidden-xs need-hidden indicator8"
      style="display:none;">1.294</td><td class="hidden-xs need-hidden indicator9"  style="display:none;">986</td><td
      class="hidden-xs need-hidden indicator10"  style="display:none;">87</td><td class="hidden-xs need-hidden
      indicator11"  style="display:none;">439403</td><td class="hidden-xs need-hidden indicator12"  style="display:none;">
      799</td><td class="hidden-xs need-hidden indicator13"  style="display:none;">7343</td>
  </tr>
  <tr class="alt">
      <td>3</td><td><div align="left">浙江大学</div></td> <td>浙江省</td><td>80</td><td class="hidden-xs need-hidden
      indicator5">88.8</td><td class="hidden-xs need-hidden indicator6"  style="display:none;">96.46%</td><td class=
      "hidden-xs need-hidden indicator7"  style="display:none;">41188</td><td class="hidden-xs need-hidden indicator8"
      style="display:none;">1.059</td><td class="hidden-xs need-hidden indicator9"  style="display:none;">803</td><td
      class="hidden-xs need-hidden indicator10"  style="display:none;">86</td><td class="hidden-xs need-hidden
      indicator11"  style="display:none;">959511</td><td class="hidden-xs need-hidden indicator12"  style="display:none;">
      833</td><td class="hidden-xs need-hidden indicator13"  style="display:none;">64392</td>
  </tr>
```

图 10.2 软科中国大学排名 2016 网页 HTML 源代码片段

```
<tr class="alt">
  <td>1</td><td><div align="left">清华大学</div></td><td>北京市</td>
  <td>95.9</td><td class="hidden-xs need-hidden indicator5">100.0</td>
  <td class="hidden-xs need-hidden indicator6" style="display:none;">
  97.90%</td>
  <td class="hidden-xs need-hidden indicator7" style="display:none;">
  37342</td>
  <td class="hidden-xs need-hidden indicator8" style="display:none;">
  1.298</td>
  <td class="hidden-xs need-hidden indicator9" style="display:none;">
  1177</td>
  <td class="hidden-xs need-hidden indicator10" style="display:none;">
  109</td>
  <td class="hidden-xs need-hidden indicator11" style="display:none;">
  1137711</td>
  <td class="hidden-xs need-hidden indicator12" style="display:none;">
  1187</td>
  <td class="hidden-xs need-hidden indicator13" style="display:none;">
  593522</td>
</tr>
```

代码中每个 td 标签包含大学排名表格的一个列数值，与表头一一对应。因此，如果要获得其中的数据，需要首先找到<tr></tr>标签，并遍历其中每个<td></td>标签，获取其值写入程序的数据结构中，这个代码封装成函数表示如下：

```
1  allUniv=[]  #存储全部表格数据，二维列表
2  def fillUnivList(soup):
3      data = soup.find_all('tr')  #找到所有 tr 标签
```

```
4      for tr in data:
5          singleUniv = []
6          ltd = tr.find_all('td')  #在每个 tr 标签中找到所有 td 标签
7          for td in ltd:
8              singleUniv.append(td.string)  #提取 td 标签中的信息
9          allUniv.append(singleUniv)data
```

上述逻辑尽管不错，却不完全。HTML 页面中除了显示大学排名的地方，其他位置也可能有表格和<tr></tr>标签，应该尽量剔除这种情况。由于爬虫针对特定网页，程序编写也不必考虑所有情况，只要能应对当前页面即可。在这个大学排名页面中，还有一处用到了表格，包含<tr>标签，但这个标签内部不包括<td>标签。因此，可以通过增加一个判断语句剔除这种情况，观察下面代码的第 6 行和第 7 行。

```
1   allUniv=[]  #存储全部表格数据，二维列表
2   def fillUnivList(soup):
3       data = soup.find_all('tr')  #找到所有 tr 标签
4       for tr in data:
5           ltd = tr.find_all('td')  #在每个 tr 标签中找到所有 td 标签
6           if len(ltd)==0:
7               continue
8           singleUniv = []
9           for td in ltd:
10              singleUniv.append(td.string)  #提取 td 标签中的信息
11          allUniv.append(singleUniv)
```

除了增加两行代码外，请将原函数的第 5 行调整为第 8 行，singleUniv=[]语句真实创建了一个空列表对象，它用于存储当前<tr>标签表示大学的数据。如果第 6 行 if 语句条件成立，说明当前读取的标签内容不是大学数据，如果创建了空列表将不再有用。因此，该语句调整到 if 语句后，只有在 if 语句判断该<tr>标签表示大学数据时才生成空列表，这样编写代码有利用占用更少的内存。

也许在其他 HTML 页面中会出现更多需要剔除的情况，而在这个大学排名网页中，仅需要剔除一种情况就能准确获得数据。大学排名爬虫完整源代码如下：

源代码 10-2：中国大学排名爬虫



实例代码 20.1　　　代码文件名 e20.1CrawUnivRanking.py

```
1   #e24.1CrawUnivRanking.py
2   import requests
3   from bs4 import BeautifulSoup
4   allUniv = []
5   def getHTMLText(url):
6       try:
7           r = requests.get(url, timeout=30)
```

```
 8         r.raise_for_status()
 9         r.encoding = 'utf-8'
10         return r.text
11     except:
12         return ""
13 def fillUnivList(soup):
14     data = soup.find_all('tr')
15     for tr in data:
16         ltd = tr.find_all('td')
17         if len(ltd)==0:
18             continue
19         singleUniv = []
20         for td in ltd:
21             singleUniv.append(td.string)
22         allUniv.append(singleUniv)
23 def printUnivList(num):
24     print("{:^4}{:^10}{:^5}{:^8}{:^10}".format("排名",\
         "学校名称","省市","总分","培养规模"))
25     for i in range(num):
26         u=allUniv[i]
27         print("{:^4}{:^10}{:^5}{:^8}{:^10}".format(u[0],\
                 u[1],u[2],u[3],u[6]))
28 def main(num):
29     url = 'http://www.zuihaodaxue.cn/\
         zuihaodaxuepaiming2016.html'
30     html = getHTMLText(url)
31     soup = BeautifulSoup(html, "html.parser")
32     fillUnivList(soup)
33     printUnivList(num)
34 main(10)
```

实例代码 20.1 运行结果如下：

```
>>>
排名  学校名称          省市      总分     培养规模
 1   清华大学          北京市    95.9     37342
 2   北京大学          北京市    82.6     36137
 3   浙江大学          浙江省    80       41188
 4   上海交通大学      上海市    78.7     40417
 5   复旦大学          上海市    70.9     25519
 6   南京大学          江苏省    66.1     20722
 7   中国科学技术大学  安徽省    65.5     18507
 8   哈尔滨工业大学    黑龙江省  63.5     25249
 9   华中科技大学      湖北省    62.9     23503
10   中山大学          广东省    62.1     23837
```

尽管实例代码 20.1 完成了中国大学排名爬虫功能，但输出效果却不尽人意，各列内容并没有对齐。这是因为中文和数字字符占用的宽度不同，利用 format()方法中的{:N}方式只是约定了输出某个变量占用的字符个数，而没有实际上约束占用的字符宽度。这是中文和西文字符混排输出时经常遇到的问题。西文字符占用一个位置宽度，而中文字符占用多个位置宽度。

实例 20 输出的每一列都有显著的类型特点，或者全是中文字符，或者全是数字。对于这类混排对齐问题，可以从填充字符角度考虑解决。以输出的第 2 列为例，实例代码 20.1 中第 20 和第 27 行约定“学校名称”列占用 10 个字符，当中文是 4 个字符时（如“清华大学”），其他 6 个字符采用西文空格填充；当中文是 6 个字符时（如“上海交通大学”），其他 4 个字符采用西文空格填充。但由于中文和西文字符占用的位置宽度不同，造成了输出不能对齐的问题。

```
24  print("{:^4}{:^10}{:^5}{:^8}{:^10}".format("排名",\
           "学校名称","省市","总分","培养规模"))
27  print("{:^4}{:^10}{:^5}{:^8}{:^10}".format(u[0],\
              u[1],u[2],u[3],u[6]))
```

解决该问题一个简单的方法是替换填充字符，采用“中文全角空格”代替默认使用的“西文半角空格”，这能够对齐中文字符出现的列。修改后的 printUnivList()函数代码如下，请用该代码替换实例代码 20.1 中的 printUnivList()函数。

```
1  def printUnivList(num):
2      print("{1:^2}{2:{0}^10}{3:{0}^6}{4:{0}^4}{5:{0}^10}".format
            \(chr(12288),"排名","学校名称","省市","总分","培养规模"))
3      for i in range(num):
4          u = allUniv[i]
5          print("{1:^4}{2:{0}^10}{3:{0}^5}{4:{0}^8.1f}{5:{0}^10}".
              \format(chr(12288),u[0],u[1],u[2],eval(u[3]),u[6]))
```

上述函数中，中文全角空格采用 chr(12288)表示，format()函数在标题栏和每行输出的参数经过手工调整。程序修改后运行输出效果如下：

```
>>>
排名　　学校名称　　　　省市　　　总分　　　培养规模
 1 　　清华大学　　　　北京市　　95.9　　　37342
 2 　　北京大学　　　　北京市　　82.6　　　36137
 3 　　浙江大学　　　　浙江省　　80.0　　　41188
 4 　上海交通大学　　　上海市　　78.7　　　40417
 5 　　复旦大学　　　　上海市　　70.9　　　25519
 6 　　南京大学　　　　江苏省　　66.1　　　20722
```

```
7    中国科学技术大学    安徽省     65.5        18507
8    哈尔滨工业大学     黑龙江省    63.5        25249
9      华中科技大学     湖北省     62.9        23503
10       中山大学      广东省     62.1        23837
```

拓展： 大学排名

世界大学排名起源于美国，从 1983 年开始每两年一次。由于排名对大学声誉和招生有非常深远的影响，大学排名逐渐成为一个产业。当前几大主流世界大学排名分别为英国《泰晤士高等教育》杂志 THE 世界大学排名、英国 QS 世界大学排名、美国 USNEWS 世界大学排名、荷兰莱顿大学世界大学排名、上海交通大学软科世界大学排名。

——这么多排名，该相信哪一个？

——角度不同、侧重不同、结果不同。要不写个爬虫抓回数据取个平均数？

思考与练习

10.7 思考实例代码 20.1 还可能有哪些改进。

10.8 修改代码输出排名后 50 位的大学。

10.9 修改代码输出某个省份的大学排名。

10.5 实例 21：搜索关键词自动提交

要点： 这是一个向搜索引擎自动提交检索关键词并获取返回结果的实例。

搜索引擎是日常工作常用的工具，也是访问互联网的门户。有时候需要自动向搜索引擎提交关键字并获得查询结果。本节以百度为例介绍搜索关键字自动提交并获得返回结果的方法。

百度搜索引擎首页为 http://www.baidu.com，当输入一个待查询关键词 keyword 时，百度程序将这个查询自动转换为链接：http://www.baidu.com/s?wd=keyword。读者可以在浏览器上手工输入这个链接，将 keyword 换成任意想查询的关键字，都能获得查询结果。

利用百度搜索提供的这个链接接口，可以通过 requests 的 get()函数提交查询，响应结果为百度搜索结果。这个问题的 IPO 描述如下。

输入：待查询关键字

处理：自动获得百度搜索结果页面，并对页面内容解析处理

输出：返回链接的标题列表

参考实例 20 的过程，首先人工分析百度查询结果页面 HTML 代码，部分片段如图 10.3 所示。由于这些 HTML 代码由机器自动生成，可读性较差，需要读者对比网页上的搜索结果和代码仔细寻找。

```
> 《<em>程序设计基础(Python语言)</em> 嵩天,黄天羽,礼欣 高等教育出... 京东</a></h3><div class="c-abstract"><span class="
newTimeFactor_before_abs m">2014年7月1日 - </span> 《<em>程序设计基础(Python语言)</em>
》系统地介绍了Pytnon程序设计语言和程序设计的基本方法。从<em>Python语言</em>发展历史、配置环境开始,详细介绍了 Pytnon语言
</div><div class="f13"><a target="_blank" href="
http://www.baidu.com/link?url=X98Iaztc9uf5P09PNDg8mlxNb0WV5sRHf8GFewJwjOcHLn9NZoXP000500YxDRZp" class="c-showurl" style=
"text-decoration:none;">item.jd.com/17309803...     </a><div class="c-tools" id="tools_8214315112877157528_2" data-tools=
'{"title":"《程序设计基础(Python语言) 嵩天,黄天羽,礼欣 高等教育出..._京东","url":"
http://www.baidu.com/link?url=X98Iaztc9uf5P09PNDg8mlxNb0WV5sRHf8GFewJwjOcHLn9NZoXP000500YxDRZp"}'><a class="c-tip-icon"><i cl
"c-icon c-icon-triangle-down-g"></i></a></div><span class="c-icons-outer"><span class="c-icons-inner"></span></span> -&
data-click="{'rsv_snapshot':'1'}" href="
http://cache.baiducontent.com/c?m=9d78d513d9810ae902b0c8690c66c0101843f7172ba1d30209d28449e3732b40501792ac51200770d3d27d1716
2102461451b38cc9f85dadcb855c249f5732676f855612a20edfcc5153b237912ae9b81897ad803084d9d6c798140dc509433cc2e78a291d099129a74b26
5f67ebb06f78fc5c762b942942b744f3f732195dc7a983&p=8b2a970293af0aff57ed9774504192&newp=c349c54ad6c04beb42a2c7710f4a83231610db2
98ffe0cc4241a1a1a3aecbf26231000d2c07a630aaf425cebf43777320634f1f689df08d2ecce7e6b9268&user=baidu&fm=sc&query=Python%D3%EF%D1
D0%F2%C9%E8%BC%C6%BB%F9%B4%A1%28%B5%DA2%B0%E6%29&qid=878cdb30000304eb&p1=2"
        target="_blank"    target="_blank"
    class="m">百度快照</a><span class="c-pingjia"> - <a href="
    http://www.baidu.com/tools?url=http%3A%2F%2Fwww.baidu.com%2Flink%3Furl%3DX98Iaztc9uf5P09PNDg8mlxNb0WV5sRHf8GFewJ
    oXP000500YxDRZp&jump=http%3A%2F%2Fkoubei.baidu.com%2Fp%2Fsentry%3Ftitle%3D%01%E3%80%8A%02%E7%A8%8B%E5%BA%8F%01%E
```

图 10.3　百度查询结果页面 HTML 代码片段

经过分析发现，页面上返回结果标题被封装在如下结构中：

```
<div…data-tools= '{"title":"…","url":"…"}'>…</div>
```

利用 beautifulsoup4 库找到 data-tools 属性值，提取带有 title 的字符串，可以看到，data-tools 内部由{}形成的数据是典型的 JSON 格式，可以用 json 库将其转换成字典，便于操作。

实例代码 21.1 给出了完整的搜索关键词自动提交程序。

源代码 10-3：百度关键词自动提交



实例代码 21.1　　e21.1AutoKeywordSearch.py

```
1   #e25.1AutoKeywordSearch.py
2   import requests
3   from bs4 import BeautifulSoup
4   import re
5   import json
6   def getKeywordResult(keyword):
7       url = 'http://www.baidu.com/s?wd='+keyword
8       try:
9           r = requests.get(url, timeout=30)
10          r.raise_for_status()
11          r.encoding = 'utf-8'
12          return r.text
13      except:
14          return ""
15  def parserLinks(html):
16      soup = BeautifulSoup(html, "html.parser")
17      links = []
18      for div in soup.find_all('div', {'data-tools':\
                                re.compile('title')}):
19          data = div.attrs['data-tools']  #获得属性值
20          d = json.loads(data)            #将属性值转换成字典
21          links.append(d['title'])        #将返回链接的题目返回
22      return links
23  def main():
```

```
24        html = getKeywordResult('Python 语言程序设计基础(第 2 版)')
25        ls = parserLinks(html)
26        count = 1
27        for i in ls:
28            print("[{:^3}]{}".format(count, i))
29            count += 1
30    main()
```

实例代码 21.1 运行结果如下，与网页返回结果一致。

```
>>>
[ 1 ]C 程序设计语言第 2 版·新版& 习题解答 - 下载频道 - CSDN.NET
[ 2 ]《程序设计基础(Python 语言) 嵩天,黄天羽,礼欣 高等教育出..._京东
[ 3 ]程序设计基础 (PYTHON 语言) PDF 电子书下载 带书签目录 samp..._微盘
[ 4 ]Python 语言程序设计_北京理工大学_中国大学 MOOC(慕课)
[ 5 ]python 基础教程(第 2 版·修订版)中文版 高清 pdf 版[30MB]..._脚本之家
[ 6 ]《程序设计基础(Python 语言)|4232695》【摘要 书评 试读】-..._京东
[ 7 ]清华大学出版社-图书详情-《Python 程序设计基础》
[ 8 ]python 基础教程(第 2 版)|python 基础教程中文高清 pdf【第..._东坡下载
[ 9 ]程序设计入门—Python - 网易云课堂
[10 ]Python 语言程序设计 python 基础教程
```

拓展：CAPTCHA 验证码

CAPTCHA 验证码是“全自动区分计算机和人类的图灵测试”（Completely Automated Public Turing test to tell Computers and Humans Apart）的缩写，它是一种区分用户是计算机程序还是人的方法，用于防止程序肆意向网络自动提交请求，如图 10.4 所示。



图 10.4 CAPTCHA 验证码示例

CAPTCHA 验证码的原理是通过图片或语音形式展现人类容易识别但程序较难识别的信息，反馈信息与生成验证码的真实信息对比能够判断输入反馈的“用户”是程序还是人类。CAPTCHA 验证码已经成为现代网络服务系统的标准配置。

在技术层面，除了搜索引擎，还可以向其他可以查询数据信息的网页提交查询关键词。正是因为有这类自动提交程序，当今开发的服务网站不得不增加图片或声

音类型的验证码，用来区分用户是计算机的自动程序还是人。技术是反映人类思想的手段，掌握了所谓“更有能力”和“更先进”的技术没什么大不了，最为可贵的是去思考如何通过技术手段为人类和世界带来更美好的未来。

能力越大、责任越大。（第一季，终）

思考与练习

10.10　仔细观察百度搜索返回页面的 HTML 代码，找到右侧“相关术语”部分对应的代码。

10.11　仔细观察并解释百度图片搜索页面的 HTML 代码。

本章小结

本章主要介绍了设计并实现网络爬虫的基本方法，结合 requests 和 beautifulsoup4 两个库的使用，讲述了如何处理 HTTP 协议以及解析网页 HTML 和 XML 页面信息的方法。本章通过中国大学排名爬虫和搜索关键词提交两个实例，展示了现代网络社会中快速抓取定向网页数据的重要价值。

程序练习题

程序练习 10-1：章节程序练习题



10.1　参考实例 24，实现按照省份输出中国大学排名的功能。

10.2　参考实例 24，实现 USNEWS 美国大学排名的爬虫，并打印结果。

10.3　编写视频网站视频播放链接的爬虫。

10.4　编写爬取 robots.txt 文件的爬虫，并分析爬取的内容。

10.5　编写根据 robots.txt 文件内容爬取网站的程序。

10.6　分析百度图片搜索返回结果的 HTML 代码，编写爬虫抓取图片并下载形成专题图片库。

10.7　编写程序测量 30 秒内成功获得百度主页的次数。

附录 A　极简计算机基础

一个有纸、笔、橡皮擦并且坚持严格行为准则的人，本质上就是一台通用计算机。

A man provided with paper, pencil, and rubber, and subject to strict discipline, is in effect a universal machine.

——艾伦·图灵（Alan Turing）

计算机科学之父、人工智能之父、数学家

学习目标

（1）了解基本的数值和数据概念。

（2）了解计算机硬件平台的组成。

（3）了解计算机软件平台的组成。

（4）了解 Internet 和 WWW 的历史。

（5）了解常用的 Python 编辑器。

计算机已经不是什么新鲜事物，计算机基础知识到处都可以找到，哪有时间看接下来的内容？作者只能冷笑。尽管计算机知识哪里都有，以计算机基础为故事情节的内容却独此一家。从 01 表示到大数据、从开关电路到 GPU 逆袭、从汇编指令到云平台虚拟化，这些不只是概念堆砌，更有内在的逻辑情节。

不看会后悔，看了更后悔，为何没早点儿看？！

A.1 数值和数据

二进制是仅由 0 和 1 组成的进位和运算逻辑，运算关系如下：

0+0=0, 0+1=1, 1+0=0, 1+1=10

二进制中，单个 0 或 1 称为比特（bit，简称 b），8 个 0 或 1 的组合称为字节（Byte，简称 B），1 个字节是 8 个比特。

由于基本电路一般包括两种状态：高电平或低电平、接通或断开，所以，为了适应并高效利用基于这种电路计算装置，计算机内部采用二进制表示数据并采用二进制运算进行数据计算。

编程语言中的整数、浮点数、字符串、列表和字典等所有数据类型在计算机内部都采用二进制表示。列表和字典这类组合数据类型由基本数据类型组合而成，整数、浮点数和字符这些基本数据类型需要计算机直接表示、存储和计算。整数采用整数的二进制形式存储在计算机中，整数的加减乘除运算采用二进制形式计算，然后将结果转换成十进制或其他进制反馈给用户。浮点数采用国际通用的浮点数算术标准（IEEE 754）表示，该标准规定了计算机如何利用二进制表示小数以及开展计算的方法。字符通过如 ASCII、UTF-8 等编码形式由一个或多个字节表示，一个字节由 8 个比特表示。

实际生活中，除了二进制、十进制外，还有很多进制方式，例如，一周 7 天是七进制，一天 24 小时是二十四进制，一小时 60 分钟是六十进制。由于二进制是计算机采用的运算逻辑，因此，编程中常用二进制、八进制、十六进制和人类熟悉的十进制。对于计算机来说，其内部对数据的表示和运算始终都是二进制，其他进制只是展示不同，原则上不需要用户人工进行进制转换。

十进制数最为常用，它使用 10 个数字符号表示，每一位只能使用 0、1、2、3、4、5、6、7、8、9 这 10 个符号中的一个，十进制数采用“逢十进一”的进位方法。

八进制使用 8 个数字符号表示，每一位只能使用 0、1、2、3、4、5、6、7 这 8 个符号中的一个，八进制数采用“逢八进一”的进位方法。

十六进制使用 16 个字符符号表示，除了数字 0 到 9，额外采用 A～F 这 6 个字符，即每一位只能使用 0、1、2、3、4、5、6、7、8、9、A、B、C、D、E、F 这 16 个符号中的一个，其中，A 表示十进制的 10，F 表示十进制的 15，只是由于十六进制的每个位置只能用一个符号表示，所以，创造性的使用了 A～F。由于字母字符 A～F 存在大小写两种形式，所以，小写 a～f 也可以用于十六进制，与大写字符含义相同。十六进制数采用“逢十六进一”的进位方法。

解决了基本的数字表示问题，计算机遇到了字符表示问题。西文字符包括英文字母、数字和各种控制符号组成，总共才 100 多个，使用一个字节（8 bit）即可表示。因此，计算机对于西文字符一般采用美国标准交换代码（ASCII）表示，ASCII 编码是一种制定于 1967 年的编码标准，能很好地解决西文字符的定义和表示问题。

ASCII 编码共包含 128 个字符，包括 26 个英文字母的大小写符号及标点符号、专用符号及控制符（如回车、换行、响铃等）。由于总数没有超过 128，所以 ASCII 码采用 7 位二进制编码，如附表 A.1 所示。其中，前 32 个通用控制符不能打印和显示，后 96 个是可以显示和打印出来的字符。通用控制符的意义或动作如附表 A.2 所示。

附表 A.1 ASCII 码表

$b_6b_5b_4$ / $b_3b_2b_1b_0$	000	001	010	011	100	101	110	111
0000	NUL	DEL	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	EXT	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	’	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	–	=	M	]	m	}
1110	SOH	RS	.	>	N	↑	n	~
1111	SI	US	/	?	O	←	o	DEL

附表 A.2 ASCII 码中通用控制符的意义或动作

符号	意义或动作	符号	意义或动作	符号	意义或动作
NUL	空	FF	走纸控制	ETB	信息组传送束
SOH	标题开始	CR	回车	CAN	作废
STX	正文开始	SO	移位输出	EM	纸尽
EXT	正文结束	SI	移位输入	SUB	减
EOT	传输结束	SP	空格	ESC	换码
ENQ	询问	DLE	数据链换码	FS	文字分隔符
ACK	承认	DC1	设备控制 1	GS	组分隔符
BEL	响铃警报	DC2	设备控制 2	RS	记录分隔符
BS	退一格	DC3	设备控制 3	US	单元分隔符
HT	横向列表	DC4	设备控制 4	DEL	删除
LF	换行	NAK	否定		
VT	垂直列表	SYN	空转同步		

由于计算机普遍使用 1 个字节作为最小的存储和处理单元，存储一个 7 位 ASCII 码只需要 1 个字节，而且还多余 1 位。因此，计算机普遍将 ASCII 码放到字节的低 7 位，最高位补零。

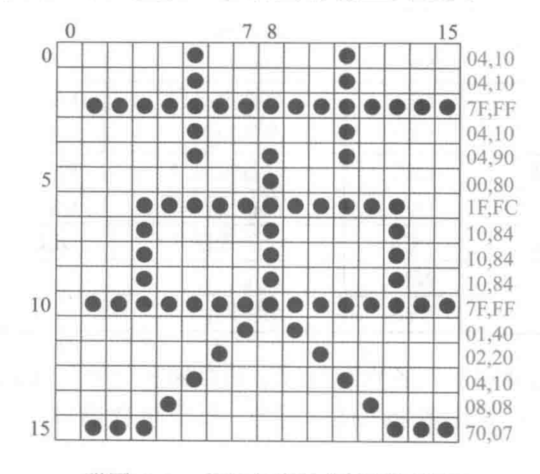
西文字符可以用 ASCII 码表示，中文汉字怎么办？不只中文，各国语言都有各自特点，所涉及字符较多，肯定无法在 1 个字节内表示。中文不是世界通用语言，计算机也不是由中国人最先发明，所以，中文字符表示问题直到计算机进入中国才开始由中国人首先研究。中国国家标准总局颁布了《信息交换用汉字编码字符集——基本集》（代号 GB 2312-80），即国标码，也称 GB2312 编码。

国标码包括 6 763 个汉字和 682 个其他基本图形字符，共计 7 445 个字符。所有国标字符组成一个 94×94 的矩阵，在该矩阵中，每一行称为一个“区”，每一列称为一个“位”。所以，该矩阵有 94 个区号（01～94）和 94 个位号（01～94）。

国标码的字符数少于 2^{16}（65 536），因此，每个汉字用 2 字节表示即可，其中每个字节仅使用低 7 位代码，最高位为 0。第一个字节表示汉字在国标字符集中的区编号，第二个字节表示汉字在国标字符集中的位编号。通过区位索引表示对应的汉字。

由于计算机的键盘一般是英文字母键盘，所以，产生了从字母组合中识别汉字的问题，于是有了输入法。汉字输入法包括区位码、首尾码、拼音码、快速首尾码、五笔字型码、电报码、仓颉码、声韵、拼形码及笔形码等。甚至只用 1～5 这 5 个数字也能输入汉字，俗称 12345 数字输入法，这是中国人的智慧，只有想不到，没有做不到。

国标码解决了汉字被计算机系统表示和存储的问题。那如何在显示器或打印机上输出汉字呢？这需要对汉字构建字形点阵图。以“英”字为例，如附图 A.1 所示，如果直接存储点阵图则需要较大的存储空间，所以，科学家设计了点阵代码。简单地说，点阵代码就是点阵图的一种表示，例如将“英”所在区域看成 16×16 的矩阵，该字第一行只有 2 个位置涂黑，所以这行 16 个位置以涂黑为 1 其余为 0 可以表示为 0x0410（十六进制形式，以 0x 开头是表示十六进制数的一种常用形式）。因此，一个汉字可以由一组这样的十六进制数表示，当需要显示汉字时，只需要在输出设备上构建一个 16×16 矩阵，然后按照这些数字的指引涂黑部分区域，即可形成汉字形状。当然，点阵规模小，分辨率差，字形不美观，但占用存储空间小，易于实现。点阵规模大，分辨率高，字形美观，但所用存储空间也大。



附图 A.1 字形点阵及点阵编码实例

计算机已经可以处理英文和中文字符了，但是，世界上还有很多其他语言，怎么办呢？计算机科学家从全球化视野角度考虑，设计了一种能够囊括世界上所有语言字符的编码方式，称为 Unicode 编码。

Unicode 是一种在国际上被广泛采用的计算机符号编码标准。对于世界上绝大多数语言所包含的文字，Unicode 都赋予它们一个唯一的编码。Unicode 是一个符号编码集合，它采用 2 个字节，即 16 位二进制对字符编码，从 0x0000 到 0xFFFF，共包括 2^{16}（65 536）个编码。例如，“语言”两个符号的编码分别为 0x8BED 和 0x8A00，“Python”6 个字母的编码分别为 0x0050、0x0079、0x0074、0x0068、0x006F 和 0x006E。

Unicode 只是规定了符号编码，并没有规定如何存储和传输这些编码。UTF-8 是一个以字节为单位的变长编码方式，用于规范 Unicode 编码的存储和使用。Unicode 编码和 UTF-8 编码的对应规则如附表 A.3 所示。

附表 A.3 Unicode 编码和 UTF-8 编码的对应规则

Unicode 符号范围（十六进制）	UTF-8 编码方式（二进制）
0000 0000～0000 007F	0xxxxxxx
0000 0080～0000 07FF	110xxxxx 10xxxxxx
0000 0800～0000 FFFF	1110xxxx 10xxxxxx 10xxxxxx
0001 0000～0010 FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

以“Python 语言”为例，两个编码的比较如附表 A.4 所示。可以看到，如果符号是英文字母，确切说是 0x7F 以内的所有符号，UTF-8 编码只使用 1 个字节表示，更加节省存储空间；对于中文字符，UTF-8 编码采用 3 个字节表示，相比 Unicode 却增加了存储空间。由于现代计算机系统的文本信息大多采用英文符号，使用 UTF-8 编码更能节省实际存储空间。

附表 A.4 “Python 语言”的 Unicode 编码和 UTF-8 编码比较

符　　号	Unicode 符号编码	UTF-8 编码
P	00 50	50
y	00 79	79
t	00 74	74
h	00 68	68
o	00 6F	6F
n	00 6E	6E
语	8B ED	E8 AF AD
言	8A 00	E8 A8 80

简单地说，Unicode 是对全球字符的统一编码，UTF-8 是对 Unicode 存储和使用的编码，将两者分开只是为了在存储西文字符时可以有效节省存储空间。要看到，计算机发展至今的绝大部分时间，计算机存储和性能都不足以支撑计算机应用，这种将 Unicode 字符统一编码和 UTF-8 存储编码分开的设计虽然在编码上看似复杂，却实实在在节省了存储空间，计算机科学家们可谓用心良苦。

至此，用 0 和 1 已经能够有效表示所有字符了，但字符不是信息。人类需要的信息该如何表示呢？

首先，需要明确，信息是数据在特定背景下的诠释。一个数据 1010，它可以是二进制数 1010，也可以是十进制数 1010，更可以是类似名字的字符串 1010，到底它是什么？严格来说，1010 只是数据，你认为它是什么信息，它就是什么信息，诠释方法由解读的人说了算。因此，计算机领域一般不用“信息”这个词，它太主观了，而用“数据”。无论是数字、音乐还是文字、视频都可以称为数据。

其次，数据分为结构化数据与非结构化数据。结构化数据一般是指存储在数据库中，可以用一维或二维表结构表示的数据。相反地，不便于用二维逻辑表来表现的数据称为非结构化数据，如图像、音频、视频等。在结构化和非结构化之间，还存在一种半结构化数据，它一般特指能够整合表示结构化和非结构化数据的数据，例如超文本标记语言（HTML）文档。

中小规模数据可以通过数据库或文件系统进行管理，无论结构化还是非结构化数据都可以用 HTML 方式通过万维网（WWW）发布在 Internet 上。一时间，数字化席卷全球，人类开始将尽可能多的信息转换成数据。新世纪伊始的 2000 年，以数字形式存储的信息只占全球数据量的 25%，其余信息通过报纸、胶片、盒式磁带等传统媒介存储。到了 2007 年，这一占比迅速超过了 90%。而之后的 2012 和 2013 两年间，互联网产生的数据已经等于人类有史以来至 2011 年所产生数据量的总和。全球数据总量呈指数级快速增长。

到了今天，数据比以往任何时候都更加深入、紧密地与人类日常活动交织在一起。各类仪器设备、传感器、网上交易、网络日志、电子邮件、视频、点击流、地理位置信息，以及现在与未来所有可被利用的其他数字化信息源产生的数据，呈现出海量、多样、复杂、纵深化与分布式的发展态势。

举个例子，一张医疗 CT 图像含有大约 150 MB 的数据，而一个基因组序列文件大小约 750 MB，一个标准的病理图与前两者相比则大得多，文件大小接近 5 GB。如果将这个数据量乘以人口数量和平均寿命，仅一个社区医院或一个中等规模制药企业就可以生成和累积数太字节（TB，约等于 10^{12}）甚至数拍字节（PB，约等于 10^{15}）数据。

可以看到，人类通过信息技术已经开始产生大量（Volume）数据，大部分数据以高速（Velocity）动态产生，包括格式化、非格式化和半格式化等多种（Variety）形式，然而，很多动态产生的数据价值（Value）较低，但也有少部分数据价值很高。大量、高速、多样和价值这 4 个特点形成了一种对数据新的描述和定义，史称“大数据”。

从 0 和 1 开始，人类最终迎来了大数据时代！这个过程中，计算机硬件以摩尔定律预测的模型在快速发展，默默地支持着人类的数据产生和处理需求。

A.2 计算机硬件组成

一个微小电路最容易实现两个状态：高电平或低电平，而高低电平可以控制电

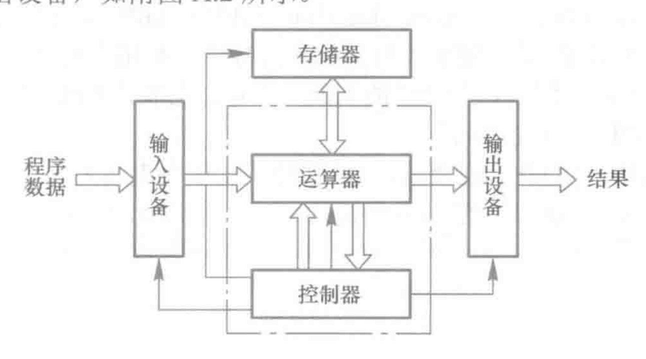
路的接通或断开。简单地说，一个电路，或高电平或低电平，就可以表示数据。

以高纯度硅为代表的半导体器件有能力受电压控制，这个物理现象促使科学家设计了 CMOS（互补金属氧化物半导体）器件，这是一种将硅和氧化物按照特定方式制作的结构，它能够根据高低电平控制电路通断。进一步，工程师将大量 CMOS 制作在一起，设计合理结构，形成了集成电路，而集成电路则具备数据计算的功能，成为了现代计算机的组成部件。单个 CMOS 及衍生设计也被称为晶体管。

从历史视角来看，半导体工业近 50 年快速发展，发展速度远超其他领域。这其中，英特尔（Intel）公司堪称是该领域的绝对引领者，没有之一。1971 年，Intel 公司推出了世界第一枚 CPU（Central Processing Unit，中央处理器），代号 4004，其中单个晶体管门电路尺寸为 10 μm（10 000 纳米），整个 CPU 包含 2 250 个晶体管，时钟频率 108 kHz。在此之前，英特尔公司创始人之一的戈登·摩尔于 1965 年先见性地提出了摩尔定律，指出单位面积集成电路可容纳晶体管的数量约每两年翻一倍。此后，该公司带领半导体工业一路发展，到 2016 年，全球半导体工业已经可以量产 14 nm 单元的集成电路，英特尔公司最新的 SkyLake 架构 CPU 采用 14 nm 晶体管门电路，单 CPU 包含超过 30 亿个晶体管，频率超过 3 GHz。CPU 诞生后的 45 年间，单晶体管尺寸下降了接近 1 000 倍，单集成电路容量提升了 1 000 万倍，频率提高了 3 万倍！

半导体技术的蓬勃发展带来了大量可用的晶体管，计算机系统结构也伴随着晶体管数量从单 CPU 结构向多 CPU 多核发展。无论 CPU 结构如何发展，现代计算机组成结构就其本质而言还是以图灵机模型为理论基础、以存储程序结构为指导思想发展而成。

艾伦·图灵提出了计算机早期的理论模型——图灵机。这是一个抽象的数学结构，但它能够按照指令方式处理数据，指导了后期计算机的设计。美籍犹太人冯·诺依曼参与了早期大型电子计算机的设计，提出了两个非常经典的设计，对计算机组织结构发展起到了深远影响。这两个设计是“存储程序结构”和“二进制编码”。根据冯·诺依曼的设想，计算机必须具有五大部件：运算器、控制器、存储器、输入设备和输出设备，如附图 A.2 所示。



附图 A.2 存储程序结构的五大部件

数据从输入设备进入计算机，控制器将运算指令从存储器中读取出来，根据指令控制运算器处理输入数据，如果产生中间结果则放到存储器中，产生最终结果由输出设备输出。这个结构简单有效，其核心是将指令和缓存数据放到存储器中，因此，该结构被命名为“存储程序结构”，也被称为冯·诺依曼结构。

在计算机的具体实现中，由于控制器和运算器很难分离，因此，它们便在一起组成了 CPU。早期 CPU 只有这两个部件，随着半导体工业提供了更多的晶体管，CPU 也包含了部分存储器，被称为高速缓冲存储器，用于加速 CPU 运算。至此，计算机有了中央运算单元，那存储器怎么组织呢？

半导体工业提供了大量晶体管，除了用于生产 CPU 外，还被用来生产存储器。用晶体管生产的存储器被当作计算机系统中的内存，辅助 CPU 运算。但由于晶体管在掉电时无法保存数据，有效输出结果不能永久保存在内存中，工程师又设计了基于磁介质的可永久保存数据的存储器，俗称为外存，包括硬盘、U 盘、磁带等。

存储设备总以字节为单位保存数据，通过地址访问字节内容。地址存放思路类似于门牌号，即为每个字节赋予一个地址，如 1010，当计算机需要访问 1010 地址的内容时，存储器找到这个字节，并将其返回。

输入设备一般采用键盘、鼠标、扫描仪、手写笔等形式，输出设备包括显示器、打印机等。随着网络的发展，网络接口成为了新的输入和输出设备。到此，五大部件及对应设备都各自发展起来，计算机成为了普通的计算设备，以个人计算机或手机等形式，走进大众生活。

CPU 有一个重要的技术指标，称为字长，指 CPU 一次能处理的二进制位数，字长总是 8 的整数倍。PC 的字长早期为 16 位、32 位，现代计算机都是 64 位（高性能计算机或服务器的字长从 16 位到 1 024 位，甚至更长）。字长不仅表示 CPU 一次处理数据的长度，也可表示参与运算数据的位数和精度。显然，用 32 位表示一个小数，如圆周率，不如 64 位表示精确。所以，64 位计算机具有更好的性能和精确度。

CPU 的通用计算性能有了保障，人类却有了更高的需求，精美游戏、3D 引擎、虚拟现实等不仅需要大量的计算资源，还需要对图像和视频有更好的显示效果。CPU 只负责计算，其硬件结构不适合绘图。从早期个人计算机开始，显示图像的任务都由图形显示卡（简称显卡）完成，早期的显卡功能还十分有限，它只是计算机中的一个扩展，核心采用一枚显示芯片。然而，显卡背后的设计公司却非常励志，这些公司充分利用了半导体工业产生的大量晶体管，设计了性能非常高的显卡芯片，被称为“图形处理器”（Graphics Processing Unit，GPU）。GPU 不仅能完美地渲染各种高清晰图像和三维效果，其高度的并行结构和计算性还被用来替代 CPU 进行大规模科学计算。部分 GPU 还集成了 CPU 的功能，大有完全取代 CPU 的发展态势。然后呢？CPU 会被 GPU 成功逆袭吗？

从半导体晶体管到 GPU 逆袭，计算机硬件在竞争中走过了近 50 年的发展历程，将人类成功送进了信息时代。学术界仍在制造下一个可能逆袭 CPU 的概念，工业界硝烟弥漫资本伺机而动，这一切预示着计算机硬件将在未来还有更值得期待的大事件。拭目以待！

A.3 计算机软件平台

直接控制 CPU 运行的指令叫机器指令。机器指令可以精细到控制一个字节数据

的移动和运算，然而，计算机系统不只有 CPU，还有输入输出设备和存储器，更大的计算机系统包括多个输入输出设备。同时，人类的计算需求也不只是完成一些简单的计算任务。简单地说，在计算机硬件之上，人类需要一个“助手”，它能更好地管理计算机，并为计算机使用者提供更加便捷友好的交流方式。这个需求产生了操作系统。

能够直接与硬件平台交流的计算机软件是操作系统。操作系统是计算机系统中最底层的软件，它控制所有在计算机中运行的程序，管理整个计算机资源，存储和调度所有数据，它是计算机硬件与应用程序及用户之间的桥梁。

操作系统能够提供更加友好的界面，使用户更为方便地使用应用软件，例如，电子制表软件、字处理器、网页浏览器和电子邮件软件等；它允许程序员利用编程语言调用其功能，也能帮助运行编程语言编写的程序。没有操作系统的硬件固然能运行程序，但也是用户的使用灾难。

操作系统是计算机系统的控制和管理中心，从用户角度来看，操作系统是用户与计算机硬件系统之间的接口；从资源管理角度看，操作系统是计算机系统资源的管理者，它的主要目的就是简单、高效、公平、有序和安全地使用这些资源。

操作系统并非与计算机硬件一起诞生，它是在人们使用计算机的过程中，为了提高资源利用率和增强计算机系统可用性而逐步发展形成的。

早期的操作系统采用命令行模式，也称控制台模式。用户调用操作系统提供的命令与操作系统交互。在 20 世纪 90 年代初，出现了带有视窗的操作系统，显著提高了用户体验，也推动了个人计算机的普及。

目前流行的操作系统主要有 Windows、UNIX、Linux、Mac OS X、Android、BSD、iOS、AIX、Windows Phone 和 z/OS 等，除了 Windows 和 z/OS 等少数操作系统，大部分操作系统都为类 UNIX 操作系统。

UNIX 是一个强大的多用户、多任务操作系统，支持多种处理器架构，按照操作系统的分类属于分时操作系统。UNIX 最早由 Ken Thompson 和 Dennis Ritchie 于 1969 年在美国 AT&T 的贝尔实验室开发。在设计开发 UNIX 操作系统的过程中，Ken 和 Dennis 发现编程语言对开发操作系统至关重要，于是他们合作设计并开发了一种用于开发 UNIX 系统的编程语言——C 语言。UNIX 操作系统和 C 语言对计算机软件的发展产生了极其重大和深远的影响，而它们却仅由两位程序员设计和完成。

Linux 操作系统是 1991 年推出的一个多用户、多任务的操作系统，与 UNIX 完全兼容，Linux 最初是由芬兰赫尔辛基大学计算机系学生 Linus Torvalds 参照 UNIX 思想开发的，实际上，他只开发了一个操作系统的内核程序，动机是兴趣和学习需要。然而，Linus 却做了一件“大事”，他把写好的 Linux 内核源代码放到了网上，供其他人下载、更新和开发。因为当时 UNIX 操作系统并不开放源代码且要支付使用费用，Linux 的开源得到了世界各地程序员的极大关注和支持，随后，在全球开源运动的浪潮下，Linux 从一个内核发展为完善的操作系统。今天，全球 70%以上在运行的服务器采用 Linux 操作系统。

Mac OS X 是苹果公司开发并在其产品中使用的操作系统。Mac OS 是首个在商用领域采用图形用户界面的操作系统，Mac OS X 是 Mac OS 的最新版本，于 2001

年首次推出。苹果公司的发展和乔布斯的故事成为了一个时代的标志和传奇。

iOS 操作系统是苹果公司为移动设备开发的操作系统，于 2007 年发布。该系统最初专为 iPhone 设计，后来陆续支持 iPod、iPad 以及 Apple TV 等苹果公司产品。iOS 与 Mac OS X 操作系统一样，同属于类 UNIX 的商业操作系统。

Android 是 iOS 系统的重要竞争对手，它由谷歌公司推出，以 Linux 为基础开发，且全系统开放源代码，主要用于便携和移动设备。在智能手机领域，除了苹果手机采用 iOS 系统，其他主要品牌手机都采用 Android 系统。

从机器指令到 PC 操作系统，从 PC 操作系统到手机操作系统，计算机软件平台在不断演进。支持移动设备后，计算机软件平台又该如何发展？

一台计算机一般只能运行一个操作系统，这是由操作系统最初的设计需求决定的。然而，计算机性能日益提高，用户产生了在一台计算机上运行多个操作系统的需求，因此，计算机软件平台进入了“虚拟化”的新阶段。

虚拟化（Virtualization）是一种资源管理技术，它将计算机的各种实体资源，如服务器、网络、内存及存储等，予以抽象、转换后呈现出来，打破实体结构间不可切割的障碍，使用户可以灵活设计并配置硬件资源，采用一个或多个操作系统管理其中的部分或全部资源。

虚拟化的一个重要呈现形式是虚拟机，它通过软件模拟具有完整功能的、运行在隔离环境中的完整计算机硬件。通过虚拟机软件，用户可以在一台物理计算机上模拟出一台或多台虚拟的计算机，这些虚拟计算机具有和真实硬件一样的部件，但能力略逊。在虚拟机内部，用户可以安装操作系统、应用程序、访问网络资源等，用户的感受与使用一台真实物理主机一样。

宿主计算机（Host PC）指物理存在的安装了虚拟机软件的计算机，这台计算机安装运行的操作系统被称作宿主操作系统（Host OS）。

客户计算机（Guest PC）指虚拟机软件中被虚拟出来的计算机，客户计算机安装的操作系统被称为客户机操作系统（Guest OS）。

虚拟机和虚拟化有多流行呢？这么说，现在所有最新的 CPU 都支持虚拟化指令，也直接支持虚拟机运行。这将是未来操作系统和计算机硬件之间一个新的软件平台。

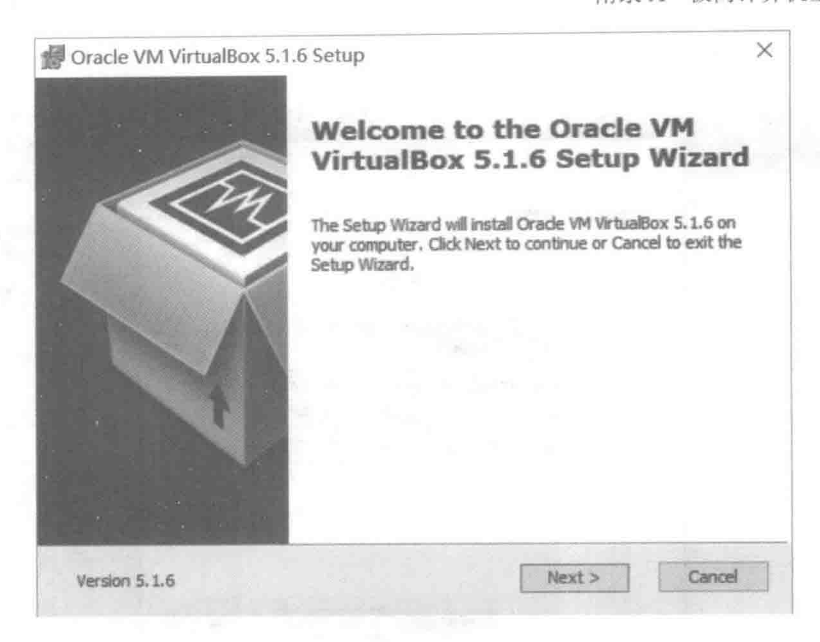
下面介绍一款经典的虚拟机软件 VirtualBox，它也是开源软件。

VirtualBox 最早由德国 InnoTek 软件公司出品，现在由甲骨文（Oracle）公司进行开发。它允许用户在 32 位或 64 位的 Windows、Solaris 及 Linux 操作系统上虚拟其他操作系统的功能。用户可以在 VirtualBox 上安装并且运行 Solaris、Windows、DOS、Linux、OS/2 Warp、OpenBSD 及 FreeBSD 等系统作为客户机操作系统。

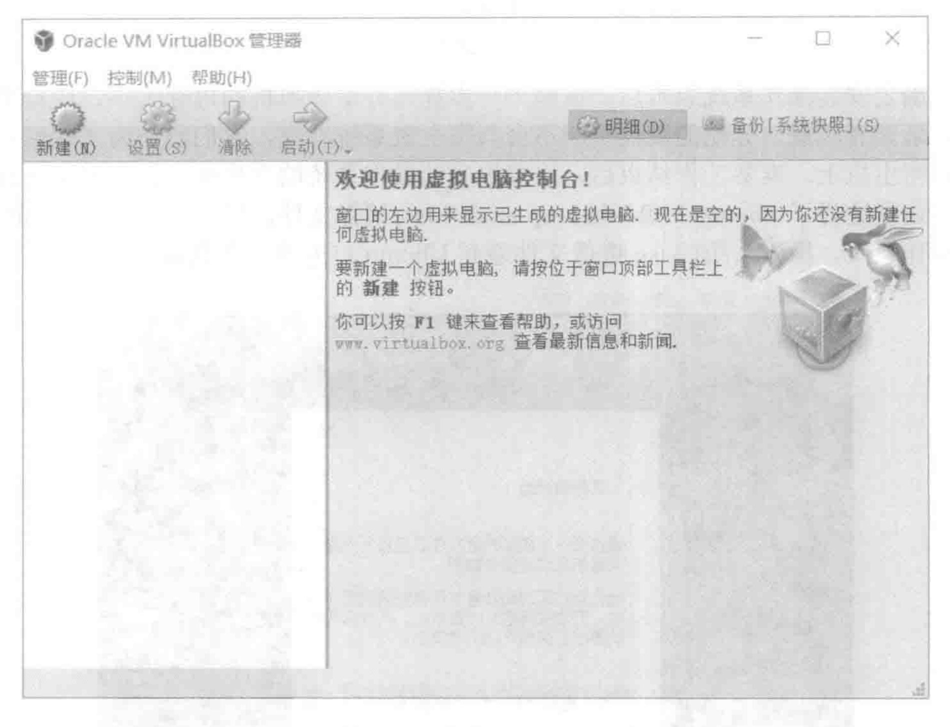
安装 VirtualBox，首先进入官方网站 https://www.virtualbox.org/，选择适合自己现有操作系统的版本下载，安装界面如附图 A.3 所示。

提示步骤很简单，安装结束后，打开 VirtualBox，如附图 A.4 所示。

通过该软件可以配置新的虚拟计算机，并为生成的虚拟计算机安装操作系统。请准备一个拟安装系统的镜像文件，本例拟安装一种 Linux 操作系统——Ubuntu 14.04。

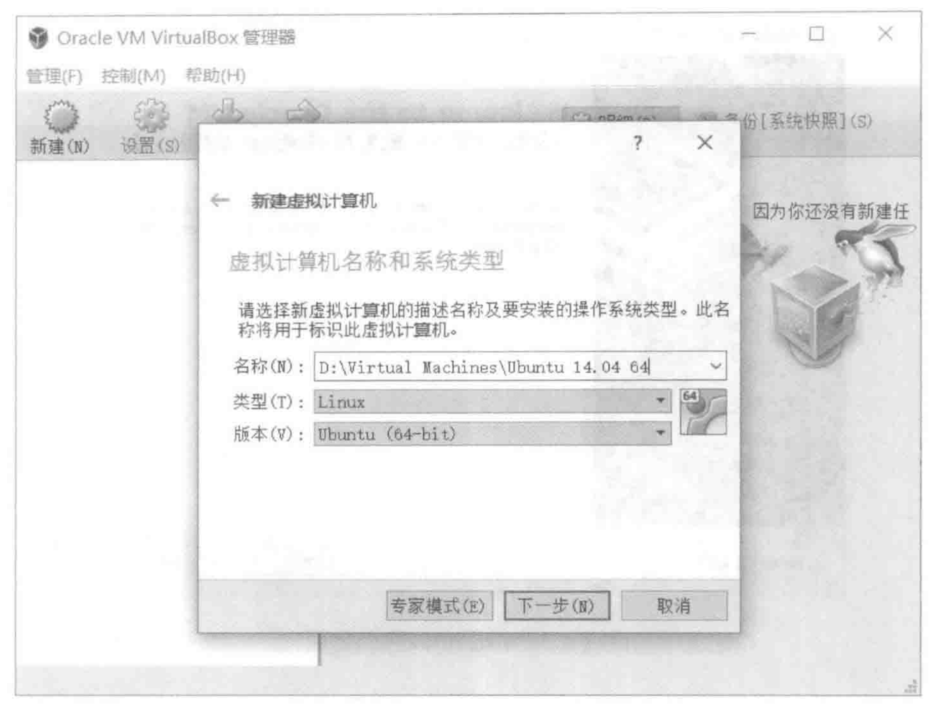


附图 A.3 VirtualBox 安装界面



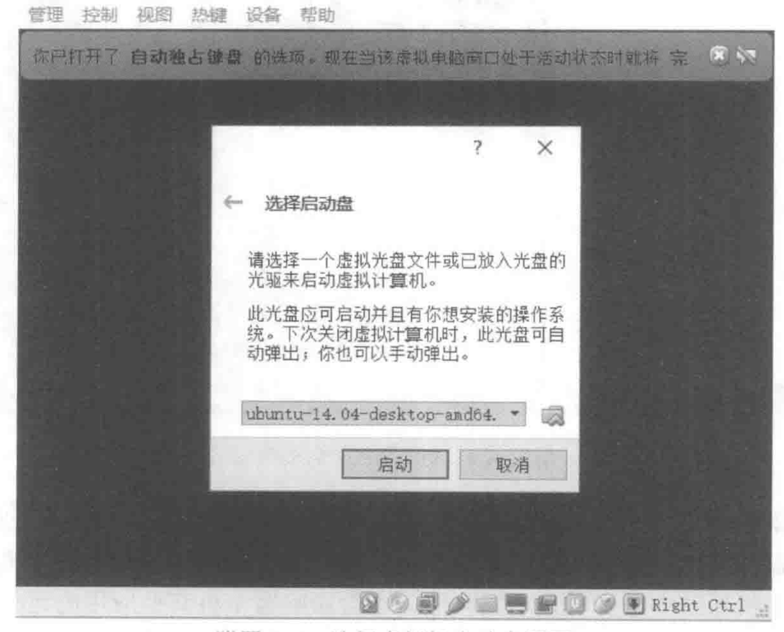
附图 A.4 VirtualBox 启动界面

通过附图 A.4 界面菜单上的“新建”按钮来新建虚拟机，并选择拟创建虚拟机存储的磁盘位置、类型、版本，如附图 A.5 所示。新建的虚拟机及虚拟机内部操作系统在宿主机系统中采用文件形式存储，不会影响宿主机其他应用的运行。可以将虚拟机理解为一个文件，用 VirtualBox 打开这个文件后，会出现一个新的操作系统。



附图 A.5 新建虚拟机界面

确定安装操作系统版本后，按照下一步提示分配虚拟机的内存大小、硬盘空间等，请读者注意，分配的硬盘空间不会与宿主机系统冲突，它们将作为文件形式保存在宿主机上。准备工作结束后，选择拟安装操作系统的文件路径，如附图 A.6 所示，这里选择了 ubuntu-14.04-desktop-amd64.iso 镜像文件，用于安装 Ubuntu 的 64 位操作系统。操作系统的 iso 镜像文件请到 Ubuntu 官方网站下载或购买。



附图 A.6 选择虚拟机启动盘界面

操作系统安装后，运行启动，客户机操作系统可以运行了。

A.4 Internet 和 WWW

网络技术可以在计算机之间传递 0 和 1，Internet 将全球计算机联网，奇迹产生了！

Internet（互联网）是网络与网络之间所连接成的庞大网络，这个网络以 TCP/IP 协议族为基础，接入设备达到几十亿，形成逻辑上的单一国际网络，Internet 是一个专用名词。互联网很有用，它提供了基本的信息传递能力，为人类创意和想象空间提供了平台。

WWW（World Wide Web，万维网）简称 Web，是 Internet 所提供的服务之一，它基于超文本链接将文本、图像、声音、视频等无缝地集成在一起，构筑成密布全球的信息资源。为了提供 WWW 服务，内容提供者需要使用 Web 服务器软件，将操作系统内指定目录的内容发布到网络上。

用户可以使用 Web 浏览器通过域名访问 WWW 所提供的内容服务，内容以页面为单位组织，超链接将页面及内部的多媒体资源链接起来，用户无须关心这些文件存放在 Internet 上的哪台计算机中，获得内容的过程由 WWW 应用利用 Internet 的连通性自动完成。

毫无疑问，本书读者都已经使用过 Web 浏览器，并体会过浏览庞大网络资源的奇妙功能。那么超链接是如何构建的呢？这里需要介绍一门专用于超链接的置标语言——HTML。

HTML（HyperText Markup Language，超文本置标语言）是一种用于创建网页的标准标记语言，常与 CSS、JavaScript 等编程语言一起用于网页前端以及移动应用界面设计。浏览器可以读取 HTML 文件，并将其渲染成可视化网页。HTML 描述了一个网站的结构语义及呈现方式，它是一种标记语言而非编程语言。HTML 元素是构建网站的基石。

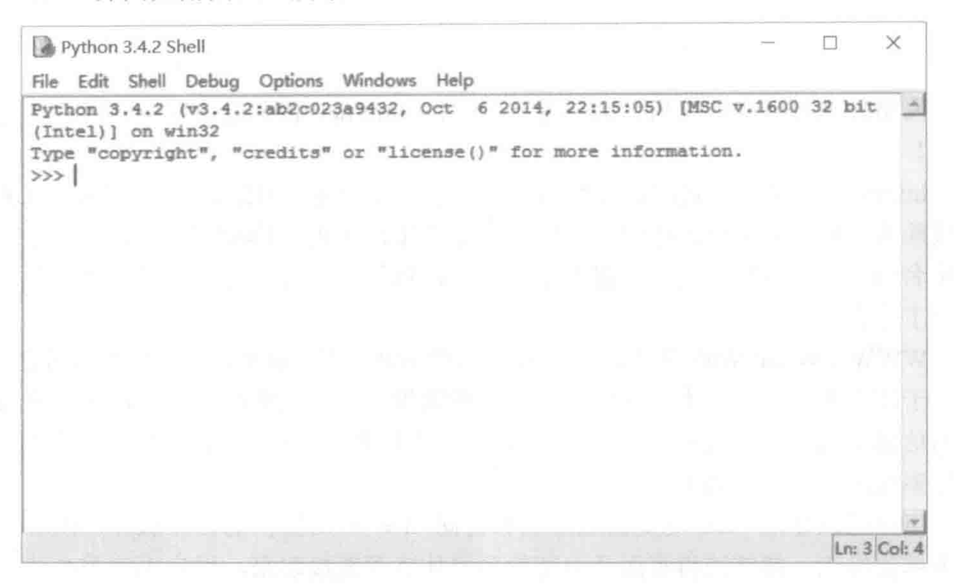
当互联技术还不成熟的时候，大家关注 Internet，因为有效地传递 0 和 1 是迫切需求。在网络技术成熟的当代，大家关注 WWW，因为有效地传播多样信息是迫切需求。关于 HTML，请读者开展扩展阅读。

A.5 常用 Python 编辑器

需要明确的是，可以采用任何编辑器编写 Python 程序，只要按照 Python 语法且保存为文件，程序都可以通过 python 命令运行。然而，功能丰富的编辑器会带来更好的编程体验。这里主要介绍 3 个编辑器：IDLE、Notepad++和 PyCharm。

IDLE 可以从 Python 官方网站下载，它是软件包自带的一个集成开发环境，建议本书读者使用这个编辑器，因为它简单易用，可以方便地创建、运行、测试和调试 Python 程序。

IDLE 界面如附图 A.7 所示。



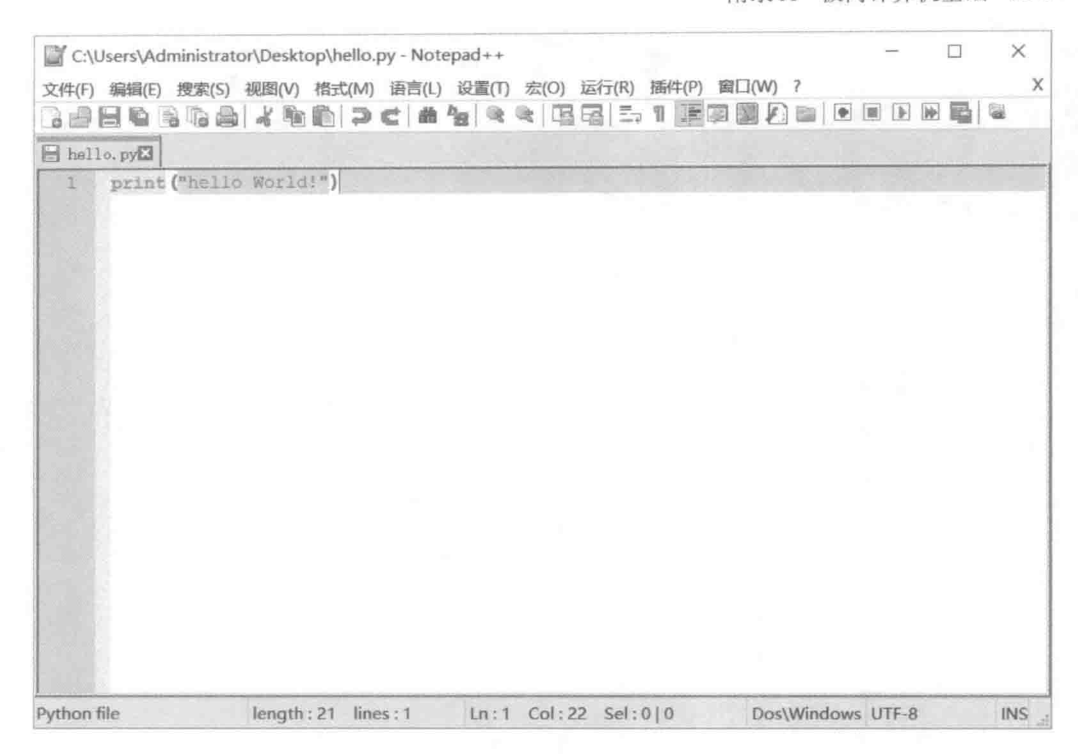
附图 A.7 Python IDLE 编辑器

从编辑器界面可以看到，系统运行了 Python 3.4.2 版本程序。尽管这不是最新的 Python 版本，但它对读者编写本书实例已经足够了。对于部分 Windows 版本操作系统，建议读者安装 3.4 或者 3.5 版本 Python 程序，而不是最新的 Python 发布程序。这是因为，最新发布程序往往带有对新功能的实验性质，而且更针对较新版本的操作系统。如果读者采用的操作系统版本不那么新，采用 3.4 版本是个明智的选择。

Notepad++是 Windows 操作系统下的一套文本编辑器，有完整的中文接口，并且支持多国语言编写的功能，最主要的是，它对编程的支持十分友好，可以支持并高亮显示多种文件类型的编程元素，如 C、C++、Python、Java、C#、XML、HTML、PHP、CSS 等。Notepad++安装包可以在 https://notepad-plus-plus.org/下载，启动后编辑 Python 程序的界面如附图 A.8 所示。

PyCharm 是由 JetBrains 公司开发的一款 Python IDE（集成开发环境），主要针对 Python 专业程序员。PyCharm 提供了非常多的编程辅助功能，比如调试、语法高亮、项目管理、代码跳转、智能提示、自动完成、单元测试、版本控制等。此外，PyCharm 还支持一些高级的 Python 第三方库，比如 Django 框架下的专业 Web 开发等。PyCharm 安装包可以从 http://www.jetbrains.com/pycharm/下载获得。

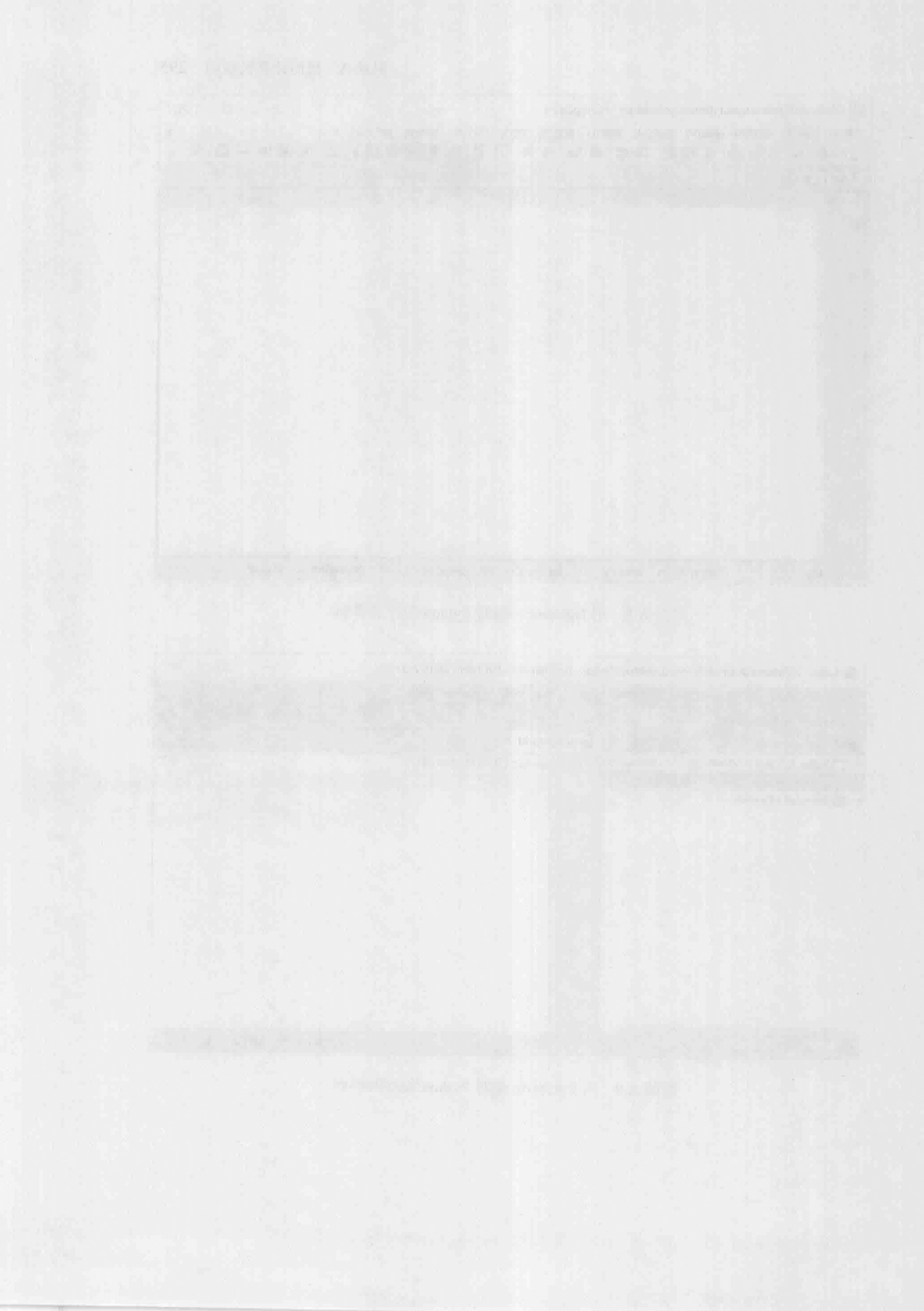
PyCharm 有两个版本：商业版和社区版，前者收费，后者免费。建议本书读者下载社区版使用。用 PyCharm 编写 Python 程序的界面如附图 A.9 所示。



附图 A.8 用 Notepad++编写 Python 程序的界面

hello - [C:\Users\Administrator\Desktop\hello] - helloworld - PyCharm 2016.2.3
File Edit View Navigate Code Refactor Run Tools VCS Window Help
hello 〉 helloworld 〉
Project
helloworld ×
hello C:\Users\Administrator\Desktop\he
helloworld
External Libraries
print("hello World!")
2:1 n/a UTF-8

附图 A.9 用 PyCharm 编写 Python 程序的界面



附录 B　人机接口和图形编程

附录 B：人机接口和图形编程



相比其他技术领域，美对于计算来说更为重要，因为软件超乎寻常的复杂，而美是对复杂性的一种终极防御。

Beauty is more important in computing than anywhere else in technology because software is so complicated. Beauty is the ultimate defence against complexity.

——大卫·盖勒特（David Gelernter）

美国艺术家、作家、耶鲁大学计算机科学系教授

学习目标

（1）了解计算机图形学的概念。

（2）掌握图形编程的基本方法及 PyQt5 库的使用。

（3）了解鼠标、键盘、文本输入框的工作原理。

（4）掌握事件的原理和信号触发机制。

（5）运用交互式编程进行图形用户界面开发。

（6）了解艺术图像的绘制方法。

从 DOS 命令行到 Windows 窗口，图形用户界面引领了计算机交互行业一次又一次革命。如今，超市的购物篮演变成了网络上的购物车，各种聊天工具的对话框逐渐替代了面对面的眼神交流，任何信息只要单击“发送”按钮就嗖的弹射出去，历史记录无论何时都忠实地保存着用户的一言一行。这些已经见怪不怪的功能蕴含了计算机交互功能的规律。

一起来了解这些规律，咱们也编个聊天界面与 QQ 来个正面对决！

本章以电子资源形式提供，请扫描侧边栏二维码获取内容。

附录 C　数据处理和挖掘

附录 C：数据处理和挖掘



我们正步入一个数据或许比软件更重要的新时代。

We're entering a new world in which data may be more important than software.

——蒂姆·奥莱利（Tim O'Reilly）

O'Reilly 媒体公司的创始人兼 CEO、Open Source 和 Web 2.0 概念的提出者

学习目标

（1）了解数据挖掘的基本概念。

（2）掌握数据挖掘的基本方法。

（3）运用第三方库实现数据挖掘的聚类、分类和回归方法。

（4）了解机器学习的基本概念。

（5）掌握一种聚类或分类方法。

数据挖掘，又译为资料探勘、数据采矿。数据挖掘就是在大量数据中寻找有意义、有价值信息的过程。当别人空谈故事时，我们要学会用数据说话。在现代社会中，与其求助于个人的信息知识储备，不如借助网络的海量信息筛选来完成特定的任务，这个特定的任务对象并不一定是枯燥的表格，它可以是一颗行星的轨道，可以是云雨的变化，或者是，一朵花？

本章将讲述如何使用 K 均值方法根据花瓣形状将花朵儿们分门别类。

本章以电子资源形式提供，请扫描侧边栏二维码获取内容。

全书快速参考索引

拓展：快速参考

快速参考（Quick Reference）是程序员编写程序时经常使用的一种以资源查找为目的的内容组织形式。快速参考往往只列出最重要的内容，不包含任何讲解，用于辅助使用者快速查找那些了解但尚未记忆的内容。本书采用1页篇幅的快速参考对应每个建议读者掌握的“能力目标”，读者可以在编程或复习时浏览这些快速参考，从而逐步达到这些能力目标。每章末的“本章小结”中将列出与能力目标同名的Python快速参考。

表　全书共11个快速参考索引

快速参考编号	快速参考名称	对应章节
参考1	Python基础语法要点	
参考2	math库	第3章3.3节
参考3	random库	第4章4.5节
参考4	datetime库	第5章5.3节
参考5	jieba库	第6章6.5节
参考6	PIL库	第7章7.2节
参考7	numpy库	第9章9.2节
参考8	matplotlib库	第9章9.4节
参考9	Requests库	第10章10.2节
参考10	Beautiful Soup库	第10章10.3节
参考11	turtle库	第2章2.3节

Python基础语法要点

库编程

```
import A
A.b()

from A import *
b() + c()

from A import b,c
b() + c()

<a>.<b>()
```

程序的分支结构

单分支结构

```
if <条件>:
    <语句块>
```

二分支结构

```
if <条件>:
    <语句块1>
else:
    <语句块2>
```

多分支结构

```
if   <条件1>:
     <语句块1>
elif <条件2>:
     <语句块2>
...
else:
     <语句块N>
```

```
<表达式1> if <条件> else <表达式 >
```

文件操作函数

```
<变量名> = open(<文件名>, <打开模式>)
<变量名>.close()
<file>.readall()
<file>.read(size=-1)
<file>.readline(size = -1)
<file>.readlines(hint=-1)
<file>.write(s)
<file>.writelines(lines)
<file>.seek(offset)
```

程序的循环结构

遍历循环

```
for <循环变量>in<遍历结构>:
    <语句块>
```

无限循环

```
while <条件>:
    <语句块>
```

字符串处理函数

```
len(x)
str(x)
chr(x)
ord(x)
hex(x)
oct(x)
```

序列类型的通用操作符和函数

```
x in s
x not in s
s + t          s * n
s[i]           s[i: j]
s[i: j: k]
min(s)         max(s)
s.index(x[, i[, j]])
s.count(x)     len(s)
```

集合类型的操作符

```
S - T
S -=T
S & T
S &=T
S ^ T
S =^T
S | T
S =|T
S <=T
S >=T
```

字典的方法或操作

```
<d>.keys()        <d>.values()
<d>.items()       <d>.get(<key>,<default>)
<d>.popitem()     <d>.pop(<key>,<default>)
<d>.clear()       del <d>[<key>]
<key> in <dict>
```

列表类型特有的函数或方法

```
ls[i] = x
ls[i: j] = lt
ls[i: j: k] = lt
del ls[i: j]
del ls[i: j: k]
s += t 或 ls.extend(lt)
ls *= n
ls.append(x)   ls.clear()
ls.copy()   ls.remove(x)
ls.insert(i, x)
ls.pop(i)
ls.reverse(x)
```

集合类型的操作函数或方法

```
S.add(x)
S.clear()
S.copy()
S.pop()
S.discard(x)
S.remove(x)
len(S)
S.isdisjoint(T)
x in S
x not in S
```

Python语言保留字 (33个)

false	elif	lambda
None	else	nonlocal
True	except	not
and	finally	or
as	for	pass
assert	from	raise
break	global	return
class	if	try
continue	import	while
def	in	with
del	is	yield

math库

引入方式 1

```
>>>import math
>>>math.ceil(10.2)
```

引入方式 2

```
>>>from math import floor
>>>floor(10.2)
```

数值表示函数

```
math.fabs(x)          math.fmod(x, y)
math.fsum([x,y,…])
math.ceil(x)          math.floor(x)
math.factorial(x)
math.gcd(a, b)
math.frepx(x)
math.ldexp(x, i)
math.modf(x)
math.trunc(x)
math.copysign(x, y)
math.isclose(a,b)
math.isfinite(x)
math.isinf(x)         math.isnan(x)
```

数字常数

```
math.pi         math.e
math.inf        math.nan
```

浮点数精确求和示例

```
>>>0.1 + 0.2 + 0.3
0.6000000000000001
>>>import math
>>>math.fsum([0.1, 0.2, 0.3])
0.6
```

幂对数函数

```
math.pow(x,y)        math.exp(x)
math.expml(x)        math.sqrt(x)
math.log1p(x)        math.log(x[,base])
math.log2(x)         math.log10(x)
```

三角运算函数

```
math.degree(x)       math.radians(x)
math.sin(x)          math.asin(x)
math.cos(x)          math.acos(x)
math.tan(x)          math.atan(x)
math.sinh(x)         math.asinh(x)
math.cosh(x)         math.acosh(x)
math.tanh(x)         math.atanh(x)
math.hypot(x,y)      math.atan2(y,x)
```

高等特殊函数

```
math.erf(x)          math.erfc(x)
math.gamma(x)        math.lgamma(x)
```

利用伽玛函数计算浮点数阶乘示例

```
>>>math.factorial(10)
3628800
>>>math.gamma(11)
3628800.0
>>>math.gamma(-11)
3628800.0
>>>math.gamma(-10.2)
-9.184935416782052e-07
```

random库

random库函数

```
seed(a=None)
getrandbits(k)
randrange(start, stop[, step])
randint(a, b)
random()
uniform(a, b)
choice(seq)
shuffle(seq)
sample()
random.setstate()
```

jieba库

常用引用方式

```
>>>import jieba
```

常用分词函数

```
jieba.cut(s)
jieba.cut(s, cut_all=True)
jieba.cut_for_search(s)
jieba.lcut(s)
jieba.lcut(s, cut_all=True)
jieba.lcut_for_search(s)
jieba.add_word(w)
```

datetime库

datetime.datetime类

属性

min	max	resolution	tzinfo
year	month	day	hour
minute		Second	microsecond

函数

```
today()
now(tz=None)
isoweekday()
isoformat(sep='T')
strftime(format)
```

strftime()格式化控制符

%Y	%a
%m	%H
%B	%I
%b	%p
%d	%M
%A	%S

turtle库

引入方式

```
>>>import turtle
>>>from turtle import *
```

控制画笔绘制状态的函数

```
pendown()         | pd()       | down()
penup()           | pu()       | up()
pensize(wid )     | width(wid)
```

控制画笔颜色和字体函数

```
color()              reset()
begin_fill()         end_fill()
filling()            clear()
screensize()
showturtle()       | st()
hideturtle()       | ht()
isvisible()
write(arg,move=False,align="left"
,font =("Arial",8,"normal") )
```

控制画笔运动的函数

```
forward(distance)  | fd(distance)
backward(distance)| bk(distance)
|back(distance)
right(angle)       | rt(angle)
left(angle)        | lt(angle)
setheading(to_angle)
position()         | pos()
goto(x,y )
setposition(x,y ) | setpos(x,y )
circle(radius,extent ,steps )
dot(size ,*color)    radians()
stamp()              speed(speed )
clearstamp(stamp_id)
clearstamps(n )      undo()
speed(speed )        heading()
towards(x,y )        distance(x,y )
xcor()               ycor()
setx(x)              sety(y)
home()               undo()
degrees(fullcircle = 360.0)
```

TurtleScreen/Screen类的函数

```
bgcolor(*args)                  getcanvas()
bgpic(picname )                 getshapes()
clearscreen()                   turtles()
resetscreen()                   window_height()
screensize(cwid ,canvh,bg )     window_width()
tracer(n ,delay )               bye()
listen(xdummy ,ydummy )         exitonclick()
onkey((fun,key)                 title(titlestring)
onkeyrelease((fun,key)          setup(wid=_CFG["wid"],h=_CFG["h"],
onkeypress(fun,key )                   startx=_CFG["leftright"],
onscreenclick(fun,btn=1,add )          starty=_CFG["topbottom"])
```

PIL库

常用引入方式

```
>>> from PIL import Image
```

处理图片时的常用属性

```
PIL.Image.format    PIL.Image.mode
PIL.Image.palette   PIL.Image.size
```

剪切，合并图像函数

```
Image.cpoy()        Image.crop(box)
Image.paste(im,box,mask)
Image.merge("RGB", (b, g, r))
```

读取，创建图像函数

```
Image.open('ilename jpg')
Image.new(mode,size,color)
Image.open(StringIO.StringIO(buffer))
TarIO.TarIO("Im.tar","Im.ppm")
Image.fromarray(obj,mode)
Image.frombytes(mode,size,data)
Image.frombuffer(mode,size,data)
Image.verify()
```

图形绘制函数

```
ImageDraw.Draw.line(ixy, fill,width)
ImageDraw.Draw.arc(xy,start,end,fill)
ImageDraw.Draw.chord(xy,start,end,fill,outline)
ImageDraw.Draw.bitmap(xy,bitmap,fill)
ImageDraw.Draw.ellipse(xy,fill,outline)
ImageDraw.Draw.point(xy,fill)
ImageDraw.Draw.polygon(xy,fill,outline)
ImageDraw.Draw.rectangle(xy,fill,outline)
ImageDraw.Draw.text(xy,text,fill,font,anchor)
ImageDraw.Draw.textsize(text,font)
```

获取数据函数

```
Image.getdata()
Image.getbands()
Image.getpixel(xy)
Image.getcolors(max=256)
Image.getextrema()
Image.getbox()
Image.histogram(mask)
```

序列操作函数

```
Image.seek(frame)
Image.tell()
```

转换，保存图像函数

```
Image.save((title, command)')
Image.show(title,command)
Image.convert(mode,matrix,colors=256)
Image.thumbnail(size,resample=1)
Image. draft()
```

像素点，通道处理函数

```
Image. point(lut, mode=None)
Image.eval(im,*args)   Image.load()
Image.split()          Image.merge()
Image.blend(im1,im2,alpha)
Image.composite(im1,im2,mask)
Image.alpha_composite(im1,im2)
```

增强，滤镜函数

```
Image.filter(ImageFilter.fuction)
ImageEnhance.enhance(Factor)
ImageEnhance.Color(im)
ImageEnhance.Contrast(im)
ImageEnhance.Brightness(im)
ImageEnhance.Sharpness(im)
```

几何变换函数

```
Image.resize(size)
Image.rotate(angle, resample=0, expand=0)
Image.transpose(method)
Image.transform(size, method, data,
resample, fill)
```

numpy库

常见引入形式

```
>>>import NumPy as np
```

创建数组函数

```
array([x,y,z], dtype=int)
arange(x,y,i)   indices(n)
linspace(x,y,n)
random.rand(m,n)
function((m,n),dtype)
ones((m,n),dtype)
empty((m,n),dtype)
```

数组属性

```
ndarray.ndim
ndarray.shape
ndarray.size
ndarray.dtype
ndarray.itemsize
ndarray.data
ndarray.flat
```

比较运算函数

```
np.equal(x1, x2 [, y])
np.not_equal(x1, x2 [, y])
np.less(x1, x2, [, y])
np.less_equal(x1, x2, [, y])
np.greater(x1, x2, [, y])
np.greater_equal(x1, x2, [, y])
np.where(condition[x,y])
```

其他运算函数

```
np.abs(x)     np.sqrt(x)
np.squre(x)   np.sign(x)
np.ceil(x)    np.floor(x)
np.rint (x[, out])
np. exp (x[, out])
np.log(x)     np.log10(x)
np.log2(x)    np.log1p(x)
```

傅里叶变换函数

```
fft(a[, n, axis])
ifft(a[, n, axis])
fft2(a[, s, axis,
   -1])
fftn(a[, s, axis])
hfft(a[, n, axis])
```

算数运算函数

```
np.add(x1, x2 [, y])
np.subtract(x1, x2 [, y])
np.multiply (x1, x2 [, y])
np.divide (x1, x2 [, y])
np.true_divide (x1, x2 [, y])
np.floor_divide (x1, x2 [, y])
np.negative(x [,y])
np.power(x1, x2 [, y])
```

矩阵运算函数

```
np.mat(data, dtype)
np.bmat ([[A, B], [C, D]])
np.eye(n)           np.dot(a,b)
np.corrcoef(a,b)  np.cov(x)
np.transpose()  ndarray.T
np.diagonal()     ndarray.H
np.trace()        ndarray.I
```

三角运算函数

```
np.sin(x)
np.cos(x)
np.tan(x)
np.arcsin(x)
np.arccos(x)
np.arctan(x)
np.degree(x)
np.radians(x)
```

统计函数

```
np.sum (x[, out])
np.mean (a, axis, dtype, out)
np.std (a, axis, dtype, out, ddof=0)
np.var (a, axis, dtype, out, ddof=0)
np.ndarray.min/max (axis, out)
np.argmin/argmax (a, axis, out)
cumsum (a, axis, dtype, out)
cumprod (a, axis, dtype, out)
```

布尔运算与位运算函数

```
np.logical_and (x1, x2[, out])
np.logical_not (x[, out])
np.logical_or (x1, x2[, out])
np.logical_xor (x1, x2[, out])
np.bitwise_and (x1, x2[, out])
np.bitwise_not (x1, x2[, out])
np.bitwise_or (x1, x2[, out])
np.bitwise_xor (x1, x2[, out])
```

形态操作函数

```
np.reshape(n,m)    np.flatten()
np.swapaxes()      np.resize(a,new_shape)
```

概率运算函数

```
np.random.normal(n,p,size)
np.random.lognormal(mean,sigma,N)
np.random.binomial(n,p,size)
np.random.hypergeometric(n1,n2,n,size)
```

matplotlib库

常见引入形式

```
>>> import matplotlib.pyplot as plt
```

绘图区域函数

```
plt.figure(figsize=None, facecolor=None)
plt.axes(rect,axisbg='w')
plt.subplot(nrows, ncols, plot_number)
plt.subplots_adjust()
```

基础图表函数

```
plt.polt(x, y,label,color,width)
plt.boxplot(data,notch,position)
plt.bar(left, height, width, bottom)
plt.barh(bottom, width, height, left)
plt.polar(theta, r)
plt.pie(data,explode)
plt.psd(x, NFFT=256,pad_to,Fs)
plt.specgram(x, NFFT=256,pad_to,F)
plt.cohere (x, y, NFFT=256, Fs)
plt.scatter()
plt.step(x,y,where)
plt.hist(x, bins, normed)
plt.contour(X, Y, Z, N)
plt.vlines()
plt.stem(x, y, linefmt,
         markerfmt,basefmt)
plt.plot_date()
plt.plotfile()
```

填充函数

```
plt.fill(x,y,c,color)
plt.fill_between(x,y1,y2,where,color)
plt.fill_betweenx(y,x1,x2,where,hold)
```

读取与显示函数

```
plt.legend()    plt.show()
plt.matshow()   plt.imshow()
plt.imsave()   plt.imread()
```

标签设置函数

```
plt.figlegend(handles,label,loc)
plt.legend()
plt.xlabel(s)
plt.ylabel(s)
plt.xticks(array,'a','b','c')
plt.yticks(array,'a','b','c')
plt.clabel(cs,v)
plt.get_figlabels()
plt.figtext(x,y,s,fontdic)
plt.title()
plt.suptitle()
plt.text(x,y,s,fontdic,withdash)
plt.annotate(note,xy,xytext,
   xycoords,textcoords,arrowprops)
```

坐标轴设置函数

```
plt.axis('v','off','equal',
         'scaled','tight','image')
plt.xlim(xmin, xmax)
plt.ylim(ymin,ymax)
plt.xscale()  plt.yscale()
plt.autoscale()
plt.text(x,y,s,fontdic,withdash)
```

Beautiful Soup库

常见引入形式

```
>>>from bs4 import BeautifulSoup
```

创建BeautifulSoup对象

```
>>> soup = BeautifulSoup(html)
>>> type(soup)
<class 'bs4.BeautifulSoup'>
```

获取tag对象

```
>>> tag = soup.head.title
>>> tag
<title>Beautiful Soup
Documentation — Beautiful Soup
4.4.0 documentation</title>
```

BeautifulSoup对象的常用属性

```
title   Strings   stripped_strings
```

Tag对象的常用属性

```
name   attrs   contents   string
```

查找tag的函数

```
Find_all( name , attrs , recursive , text , **kwargs )
find( name , attrs , recursive , text , **kwargs )
```

Tag中数据的获取函数

```
get(attribute)
get_text()
```

Requests库

常见引入形式

```
>>>import requests
```

各种请求方式函数

```
requests.get(url)
requests.post(url, data = {'key': 'value'})
requests.delete(url)
requests.head(url)
requests.options(url)
requests.put(url, data = {'key': 'value'})
```

常用属性

```
requests.status_code
requests.text
requests.encoding
requests.content
requests.raw
```

常用函数

```
requests.json()
requests.raise_for_status()
```

参 考 文 献

[1] Guido van Rossum, Fred L. Drake Jr. The Python Language Reference (release 3.5.2)[M]. Python Software Foundation, 2016.

[2] John Zelle. Python Programming: An Introduction to Computer Science 2nd Edition [M]. Oregon: Franklin, Beedle & Associates Inc., 2010.

[3] http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html.

[4] Harrison C., Amento B., Kuznetsov S., Bell R. Rethinking the Progress Bar[C]. In Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology, Rhode Island, USA, 2007.

[General Information]
书名=Python语言程序设计基础（第2版）
作者=嵩天，礼欣，黄天羽著
页数=311
SS号=14207571
DX号=
出版日期=2017.02
出版社=高等教育出版社